BACHELOR OF ENGINEERING DEGREE WITH
HONOURS IN

Digital Systems & Computer Engineering

Final Year Project Report

School of Electronic, Communication and Electrical Engineering

University of Hertfordshire

Maze Solving by a Humanoid Robot

Report by
Nauman Shah

Report For
Project Supervisor: **Dr. David Lee**

Date
20th April 2009

DECLARATION STATEMENT

I certify that the work submitted is my own and that any material derived or quoted from the published or unpublished work of other persons has been duly acknowledged (ref. UPR AS/C/6.1, Appendix I, Section 2 – Section on cheating and plagiarism)

Student Full Name: [Nauman Shah]

Student Registration Number: [07155081]

Signed: ……

Date: 20 April 2009

ABSTRACT

This report describes the work undertaken in developing software to successfully make a humanoid robot to walk through a simple maze, avoiding any obstacles on its way. The robot used was built using the Bioloid robot construction kit by Robotis. The software written allows the robot to play different motion sequences which are in turn used to implement a simple wall following algorithm.

The report starts with an introduction to the project aims and objectives followed by a detailed description of the subject background explaining the hardware and software used. The report explains the procedure undergone in order to achieve the aim and details the outcomes achieved. The report concludes with a brief overview of the project and future development recommendations.

ACKNOWLEDGEMENTS

Above all, I would like to thank the Almighty for His endless blessings for giving me the knowledge, strength and the opportunity to be where I am. It is only through His grace that this achievement could truly be accomplished.

Secondly, I would like to thank my supervisor Dr. David Lee for the patient guidance, encouragement and continuous support he has provided. I have been extremely lucky to have a supervisor who cared so much about my work and inspired me with a positive outlook at every stage. I would like to express my deep gratitude to him for his invaluable support and time.

I would like to thank my friends, Ramiz Ahmed and Talha Khalid for their help and support throughout the period of the project.

Last but not the least; I would like to thank my family for their generous and never-ending support and for teaching me the basic principles of life.

TABLE OF CONTENTS

DECLARATION STATEMENT i
ABSTRACT i
ACKNOWLEDGEMENTS i
TABLE OF CONTENTS ii
LIST OF FIGURES iv
GLOSSARY 1
1. Introduction 2
1.1 Introduction 2
1.2 Aims and Objectives 2
1.2.1 Project Aim 2
1.2.2 Objectives 3
1.3 Methodology 3
1.4 Feasibility Study 4
1.5 Project Cost Analysis 4
1.6 Product Marketability 5
1.7 Summary 5
2. Subject Overview 6
2.1. Robots 6
2.1.1. Humanoid Robots 6
2.1.2. History of Humanoid Robots and State of Art 6
2.2. The Bioloid Comprehensive Kit 8
2.2.1. Hardware 8
2.2.2. Software: 12
2.2.3. Communications 17
2.3 Chapter Summary 19
3. Description and Discussion of Work Done 20
3.1. The Bioloid Humanoid Model 20
3.2. WinAVR Compiler 21
3.3. Design and Implementation 21
3.3.1. Example code functions 22
3.3.2. Servo Functions 24
3.3.3. Motion Editor Functions 25
3.3.4. Sensor Functions 31
3.3.5. Maze Solving 33
3.4. Testing and Debugging 35
3.5. Outcomes 36
3.6. Chapter Summary 37

4. Conclusion and Other Aspects 37
4.1. Conclusion 37
4.2. Review of Objectives 38
4.3. Time Management 39
4.4. Further Development 39
4.5. Chapter Summary 39
REFERENCES 41
BIBLIOGRAPHY 44
APPENDICES 45
Appendix A – Project Time Plan 46
Appendix B – AX-12 Servo Control Table 47
Appendix C – AX-S1 Sensor Control Table 48
Appendix D – Instruction Set for Dynamixel 49
Appendix E – Software Listings 50
Appendix F – Instruction guide to download a C program on the CM-5 78

LIST OF FIGURES

Figure 1 - The Bioloid Comprehensive Kit [14] 8
Figure 2 - Dynamixel AX-12+ Servo [11] 8
Figure 3 - AX-12 Servo Movement Range [9] 9
Figure 4 - AX-S1 Sensor Module [18] 9
Figure 5 - AX-S1 Module sensor and receiver location [13] 9
Figure 7 - CM-5 Module Top View [14] 11
Figure 6 - The CM-5 Main Controller Module [18] 11
Figure 8 - Bioloid Frame Set [14] 12
Figure 9 - Behavior Control Screenshot 12
Figure 10 - Behavior Control program example [14] 13
Figure 11 - Motion Editor Software Screenshot 13
Figure 12 - Motion Editor Screen Description 14
Figure 13 - Bioloid's flash memory division 15
Figure 14 - Robot Terminal Screenshot [18] 16
Figure 15 - Wiring of CM-5 Module with Dynamixels [9] 17
Figure 16 - CM-5 Internal Circuit Diagram [9] 17
Figure 17 - Bioloid Communication Protocol 18
Figure 18 - Bioloid Humanoid Model [22] 20
Figure 19 - WinAVR Comiler Screenshot [17] 21
Figure 20 - Bioloid Humanoid Sensor Directions 33
Figure 21 - Right-hand wall following example [21] 33
Figure 22 - Flowchart representing the 'decision making' wall following algorithm developed... 34
Figure 23 – Maze map and robot's route 36
Figure 24 - Robot solving the maze 36

GLOSSARY

Term	Meaning
Dynamixel:	This is the basic unit of the Bioloid which acts as a joint or a sensor.
TTL:	Time to live
CAD:	Computer Aided Design
FIRA:	Federation of International Robo-Soccer Association)
DOF:	Degree of Freedom
GUI:	Graphical User Interface
LED:	Light Emitting Diode

1. Introduction

This chapter gives a brief introduction to the project highlighting the main aim of the project and objectives set to achieve the aim. The chapter explains the methodologies implemented to accomplish the objectives. It, then, talks about the feasibility study carried out and the cost analysis of the project. It talks about the project's marketability and its applications in the practical world.

1.1 Introduction

In 1923, Karel Capek, a Czech writer, coined the term 'robot' in his play Rossum's Universal Robots, as a derivative of the word 'robota', meaning 'serf labour'. Robots today, built for tedious and dangerous for human tasks, are still far from achieving the fictional human-like intelligence and physical capabilities depicted in the play. Various companies all over the world have devoted fortunes for research in the field and are working hard to make robots one step closer to their fictional counterparts and with the show of effective results, are gaining the increasing interest of many.

Even though building a humanlike robot requires a combination of complex hardware and software potentials but with the technological facilities provided by companies like Robotis, the subject can be pursued by educational institutes at a lower, simpler and cheaper scale. The project detailed by this report is based on such a task of building a humanoid robot able to walk, avoid obstacles on its way and solve a maze.

1.2 Aims and Objectives

1.2.1 Project Aim

The main aim of the project was to design and implement a program in C programming language which allowed the Bioloid robot to walk its way out of a simple maze.

The software was to be developed using C programming language correlating it with the existing software packages available with the Bioloid Development Kit in order to achieve an optimum package, utilising both, the user-friendly GUI of the provided software, the Motion Editor, (to easily create motion sequences for the robot), and to gain the complete control over the robot using C programming language.

1.2.2 Objectives

In order to achieve the main aim, it was broken down into a number of objectives for each of which a separate work stage was designated. The different objectives have been listed below along with the details of why the objectives were chosen and the potential benefits their completion will provide:

i. **Research and study the hardware components, existing software development kits and the communication protocols.**
Through this objective, extensive knowledge about the basic concepts and background of robotics will be gained. It will highlight the basic operations, design and architecture of the hardware used to build the robot.

ii. **Devise functions in C programming language for the main controller module of the robot, to send simple commands to servos and read information from the servos and the sensor.**
This will enable the main controller module of the robot to read and write data to the servos, allowing the ability to send a servo to a desired target position at a specified speed and monitor its current position

iii. **Implement functions in C programming language for the main controller module of the robot, to read the motion data stored in the flash memory.**
This objective will allow motion sequences, for the movement of the robot, to be created using the software development kit provided and play these motions on the robot using the functions created before.

iv. **Implement a simple wall following algorithm to make the robot find its way out of a simple maze.**
This objective will enable the robot to find its way out of a simple maze by executing a simple maze solving strategy.

1.3 Methodology

In order to achieve the objectives, certain steps were carried out.

Beginning with the literature research, the documentation provided with the Bioloid Comprehensive Kit were thoroughly studied and a comprehensive study was done to discover new ways to program the robot besides using the software development kit provided.

The example C language code provided in the Bioloid manual was studied and some basic functions were extracted from it.

Further functions were implemented using the WinAVR compiler, by AVRFreaks, coupled with the Programmer's Notepad.

The Motion Editor software was used to create walking and turning motion sequences, which were downloaded on to the flash memory of the main controller module of the robot, using the Robot Terminal software provided. The functions developed were then used to implement these motions in sequence to implement wall-following and were tested in a simple maze created with cardboard pieces.

1.4 Feasibility Study

Before the implementation of the project work was done, a feasibility study was carried out and a report was generated to establish an understanding between the student and the supervisor on the formal title of the project as well as the objectives and goals set out to be achieved.

The report looked in to the 'aims and objectives' set out for the project and gave a brief overview of the entire project giving a short description of the apparatus used. It explained the various stages of work involved in order to achieve the required goal.

A detailed action plan was projected showing how the project work is to be undertaken together with planned time scales for the successful completion of the project. The project time plan is attached in the Appendix A.

1.5 Project Cost Analysis

The project cost was mainly based on the Bioloid Comprehensive Kit itself, which was provided by the university. The Bioloid Comprehensive Kit is available in the market at a current price of £625.31. [1]

The software development kit provided with the Kit is freeware and hence can be installed on as many computers as per required.

The compiler used was a freeware GNU GCC compiler WinAVR coupled with Programmer's Notepad, by AVRFreaks.

1.6 Product Marketability

Even though there are no direct applications of the project carried out in the market, it is an excellent educational platform and can be used for various research purposes. Famous for its flexibility in the mechanical design of the robot and its low cost, many educational institutes are using the Bioloid platform for engineering education. The easily programmable, modular based design of the kit can allow students to understand the concepts of robotics and mechanical engineering. [2]

The platform is also known to be very popular in the international robotics competitions such FIRA (Federation of International Robo-Soccer Association). [3]

Also, the Bioloid platform is currently being used by the U.S. Naval Academy in their Mechanical Engineering courses. [4]

1.7 Summary

This chapter gave a brief introduction to the project highlighting the main aim of the project and detailed the objectives set to achieve them. It looked in to the methodologies undertaken to accomplish each aim of the project. It talked about the feasibility study carried out and the cost analysis of the project. The chapter ends by talking about the projects marketability, its importance in the market and the applications in the practical world.
The next chapter will talk about the Bioloid Kit and explain its hardware and software operations.

2. Subject Overview

This chapter details the background knowledge of the project. It starts with giving a brief introduction to the field of Robotics. It talks about the importance of humanoid robots and their development history in the industry.
It then goes on to explain in the detail the architecture and operation of the hardware and the design of the software used in the project. It also talks about the communication protocols of the system explaining how data is transferred amongst the hardware and between the hardware and software.

2.1. Robots

Robot, according to Hans Moravec - a robotics expert, is a mere machine to transcendent human mind. For ages, the human mind itself has been under the microscope of scientists who have done immeasurable struggles to build a machine capable of achieving human intelligence and physical capabilities. To some extents they have been successful, to others they soon will be. Some even say that robots will soon surpass humans. Today, a number of different types of robots have been developed and are being used in industries, as well as in homes to replace human efforts often without a lot of human intervention. Suitable for repetitive, precision demanding or even tasks dangerous for humans, robots have been, and even more will be, an essential innovation serving the mankind in incomprehensible ways.

2.1.1. Humanoid Robots

The field of autonomous robotics is rapidly growing and attracts an accelerating interest. Although the various robot model designs have been designed in the field of robotics, be it hobby or commercial, the holy grail has been the production of the humanoid biped model. The humanoid model not only has the overall appearance resembling that of a human body; it also is designed to imitate the human behaviour and motion capabilities. Its autonomous nature allows it to adjust strategies based on the surroundings and adapt to new surroundings, hence requiring high performance in both software and hardware. It is argued that if these requirements and successfully fulfilled, humanoid robots could be the next dominating mechanical industry. [5] [6]

2.1.2. History of Humanoid Robots and State of Art

Since the launch of the first human-size biped humanoid, '**Wabian**', by the Waseda University in Japan in 1995, many humanoid robots have been built in both industry and academia for variety of purposes. Many companies, such as Honda, Sony, Samsung, Toyota and Fujitsu have been working for year on the development of humanoid robots. Honda after developing

a couple of prototype models finally launched its well known humanoid robot '**ASIMO**', in 2000. ASIMO was followed by Sony's small humanoid robot named '**QRIO**' and Fujitsu's commercial robot '**HOAP**' in 2001. Mainly, these robots were used for exhibition and advertisement purposes but some were also sold for research purposes to universities. Other companies like Kondo Kagaku, introduced its programmable biped humanoid in June 2004 followed by Samsung's humanoid models name **MAHRU** (male) and **AHRA** (female), in November 2005. Mitsubishi Heavy Industries, also developed a domestic robot named '**Wakamaru**' primarily intended to provided companionship to elder and disabled people. Another research platform **HRP-1** developed by AIST (National Institute of Advanced Industrial Science and Technology). In addition to biped walking, most of these robots also exhibited some human characteristics, for example the **HRP-4C**, an advanced model of the HRP-1, launched in 2009, is capable of voice recognition and speech synthesis.
In academia, many institutes like University of Tokyo, Waseda University, University of Munich, Beijing University and many others developed their very own humanoid robots for robotics research. [6] [7]

Although some of these projects were sold to various universities for research and educational purposes, due to the high price, they were well out of reach of hobbyists and even some universities. Taking advantage of the opportunity of the high demand by robot enthusiasts, Kondo, a Japanese company, developed and launched the first biped humanoid robot kit, the **KHR-1**, allowing consumers to design, build and program robots at a smaller and cheaper scale. With it being a great success, other companies such as Hitec with its **RobotNova 1** and Robotis with its **Bioloid Edutainment Kit** expanded the market. These development kits, besides their remarkable educational and research application, supported a new field which resulted in a whole community of robot enthusiasts and students all over the world designing and building their very own robots. Global events such as Robo-One, FIRA (Federation of International Robot-soccer Association) and Wonderful Robot Carnival, allows these students and hobbyists from all over the world to present and compete their robots against each others. [8]

2.2. The Bioloid Comprehensive Kit

The project detailed by this report was based on one of such, earlier discussed, development kits; The **Bioloid Comprehensive Kit** by a Korean company Robotis. The Bioloid Comprehensive Kit is the first of its kind to be built around smart serially controlled servo technology. This allows users to construct a wide variety of autonomous robots and configure them using one controller board. The kit includes 18 Dynamixel series AX-12 servos (the Robotis servos), 1 Dynamixel AX-S1 sensor module, a CM5-processor module (the main controller) and a number of nuts and bolts.

Figure 1 - The Bioloid Comprehensive Kit [14]

The kit comes with three software utilities to program the robot. A GUI based, **Motion Editor** which allows the users to create 3D motion sequences and play them on your robot. **Behavior Control Program** on the other hand, lets you write algorithms in modular forms and design the logic for the robot. Last, a **Robot Terminal** software is a command based application for low-level management of the robot.

2.2.1. Hardware

The modular based kit has 3 main building blocks in addition to the simple assembly parts:

i. AX-12 Servo Module

Figure 2 - Dynamixel AX-12+ Servo [11]

The Dynamixel series AX-12+ servo, from Robotis, is one of the most advanced actuators available in the market. It is a smart, modular servo that incorporates a gear reducer, a precision DC motor and a control circuitry with networking functionality, all in a single package [9]. The powerful servo, despite its compact size, can produce high torque and provides the necessary strength to withstand large external forces due to its high quality structural specifications.

Each AX-12 actuator is equipped with a sensor module which enables it to detect and act upon environmental and internal conditions such as changes in temperature or supply voltage, which are all handled by the actuator's own built-in microcontroller. An alarming system and a status LED alerts the user when parameters deviate from user defined ranges.

The servo also has the ability to track and report back its speed, shaft position and load placed on it, which allows functionality to constantly monitor the device. These parameters can be accessed through a single-bus, high-speed, serial communication protocol. The bus is used for both data and power supply and hence can be used to daisy-chain several servos together allowing the construction of an extendable network of motors which can be individually accessed and controlled by the main controller module.

Each servo can move to a 300 degree range in 1024 increments, as shown in the figure 3, or can be set in continuous mode for full rotation. The speed and maximum torque can also be controlled by their respective parameters ranging from 0-1023.

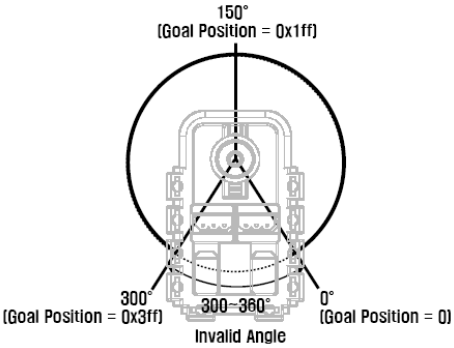


Figure 3 - AX-12 Servo Movement Range [9]

Appendix- B includes the control table containing the information on the status and the operation of the Dynamixel servo. The servo is operated by writing values in the appropriate address of the required parameter and the status is read by reading the values off the appropriate address. [10] [11]

ii. The AX-S1 Sensor Module

The AX-S1 sensor module, by Robotis, is compact, yet smart, sensor module which allows the functionality to detect light, measure distance, receive and transmit infrared remote control signals and detect sound. Not only that, it also has built-in buzzer which allows the playback of musical notes and other special note effects. The casing of the AX-S1 sensor module is very similar to that of the AX-12 servo module, hence is also made with high quality materials which provide the necessary resilience to endure large external forces.

Figure 4 - AX-S1 Sensor Module [18]

Like the servo module, this module too detects subtle changes such as the internal temperature and the input voltage, in response to which the module's microcontroller adapts different strategies. An alarming system detects and alerts the user of any out of range values of the internal temperature and voltage.

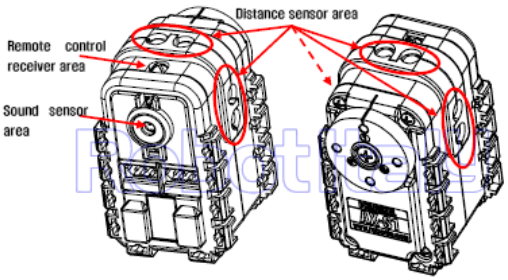


Figure 5 - AX-S1 Module sensor and receiver location [13]

The module is embedded with 3 Infrared sensors mounted on the front, left and the right side of the module (as shown in Figure 5), making it possible to sense from all three directions. Each of these sensors allows two types of measurements; distance and luminosity sensing. The former is measured by emitting a high frequency Infrared, by an integrated emitter, and measuring the amount of the rays return after bouncing off an object. The luminosity is measured using the similar technological concept but without the emitting source. Thus, this sensor only measures the amount of infrared emitted from a source of illumination, sensing the brightness of the light. These values can range from 0-1023, and the input is compared to a preset 'compare value', setting or resetting the relevant flags, Obstacle Detection Flag or Luminosity Detection Flag, accordingly. Refer to the table below for bit representation of each sensor flag.

Bit	Representation
Bit 2	An object is detected on the Right Sensor /Light Detected
Bit 1	An object is detected on the Center Sensor /Light Detected
Bit 0	An object is detected on the Left Sensor /Light Detected

Table 1 - AX-S1 Sensor Obstacle Detection Flag Description

The AX-S1 also has some other features that will not be discussed in detail since they are not relevant to the project. The features include a built-in remote control sensor, which makes it possible to send and receive Infrared data between sensor modules. There is also an internal microphone, which allows it to detect the current sound levels and maximum loudness making it possible to count the number of sound instances that has occurred. The built-in piezo-electric buzzer can play musical notes and produce beeping sounds.

Appendix - C contains control table containing the control information on the status and the operation of the Dynamixel sensor. The sensor is operated by writing values in the appropriate address of the required parameter and the status is read by reading the values off the appropriate address. [12] [13]

iii. The CM-5 Controller Module

The CM-5 module, shown in Figure 6, is the brain and the heart of the Bioloid robotics system. The module, based around the powerful and flexible Atmel Atmega 128 microcontroller, offers the functionality to store and execute programs to control the Dynamixel sensor and servos. The module communicates with the Dynamixels via a TTL Serial Network, through its four Dynamixel serial ports. The figure shows the 3-wire-dropped serial network. The CM-5 communicates with the servos and the sensors, via this network, using their unique Network ID's programmed in their non-volatile memory. The Bioloid is connected with the PC using a serial connection via a 'Serial Cable Jack'.

Figure 6 - The CM-5 Main Controller Module [18]

The CM-5 features 4 programmable control buttons, a power switch, a 'Start' button and a 'Mode' button (shown in Figure 7). The Start button starts the program loaded into the memory of the CM-5 and the Mode button switches between the operating modes of the CM-5 Module.

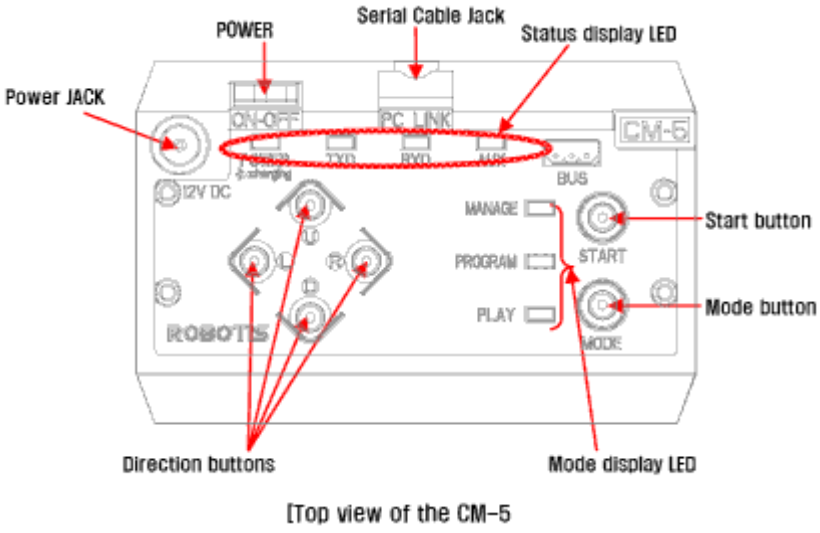


Figure 7 - CM-5 Module Top View [14]

The controller operates in 3 different modes:

1. Manage Mode: This mode is used when the user wants to monitor the status of the CM-5 unit or the Dynamixels, or when a motion is to be tested.
2. Program Mode: This mode is used for editing the motion of the robot.
3. Play Mode: This motion is used to run the program loaded in the module.

The firmware of the CM-5 module, provided with the Bioloid Kit, allows the charging of the module using the SMPS (switched-mode power supply) connected through the power jack.

The CM-5 module is housed in a sturdy plastic casing which houses the controller as well as the battery pack. [14] [15] [16]

iv. Assembly Parts

Figure 8 - Bioloid Frame Set [14]

The frames provided with the Bioloid Kit include frames made from injection molded plastic and are designed to firmly hold the Dynamixels together and with the CM-5 controller. These frames are designed excellently to allow configuration of various different robot models.
There are also a number of bolts and screws provided to connect these frames together.

2.2.2. Software:

The Bioloid Comprehensive Kit is supplied with three different software packages:

i. The Behavior Control

The Behavior Control Program is a simple yet efficient program which allows the user to design simple flowchart structures telling the robot what actions to take for a given state. The icon based interface allows the functionalities accessible within the CM-5 include playing the motion sequences, external control buttons and using the CM-5's timer. It has an intuitive modular GUI which enables the programmers to create logical statements which controls the behavior of the robot according to the input from the sensors.

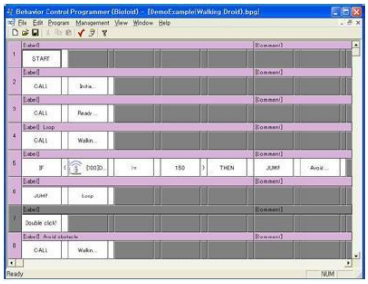


Figure 9 - Behavior Control Screenshot

The graphical sequencer interface allows the ability to define rules and link them with a defined action or with a series of motions, created in the Motion Editor Software (discussed later), within a loop or a branch statements in order to implement the required algorithm.

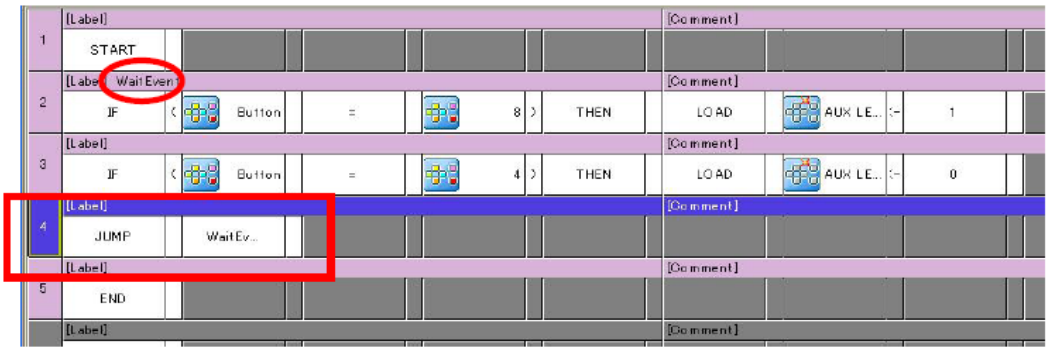


Figure 10 - Behavior Control program example [14]

The figure shows an example program running in a simple loop, which turns the auxiliary LED on or off depending on the input state of the control buttons. [14]

ii. Motion Editor

The Motion Editor software allows the user to create unique motion sequences for your robot through CAD rendering. The 3D GUI interface allows you to monitor and finely tune the servo positions by simply incrementing, decrementing or even entering the desired target position. Not only that, the software also allows the ability to set the position of the servos manually by hand. Once all the servos of the robot are in the desired position, each servo's positions, a motion instance, can be saved as a motion '**Pose**'. The motion pose can then be saved and executed whenever called. Several of these poses can then be called in a sequence to make up a short motion, called a **'Page'**. 7 poses make up a single page. The motion Pages are downloaded in the flash memory of the CM-5 using the Behavior Control or the Robot Terminal software and further be called, using their specific page number, in a sequence of their own to make the robot perform more complex motions.

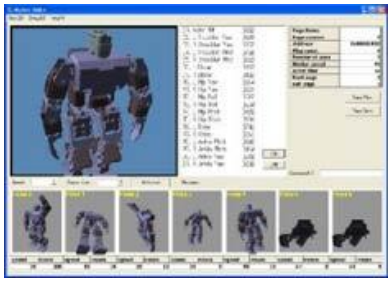


Figure 11 - Motion Editor Software Screenshot

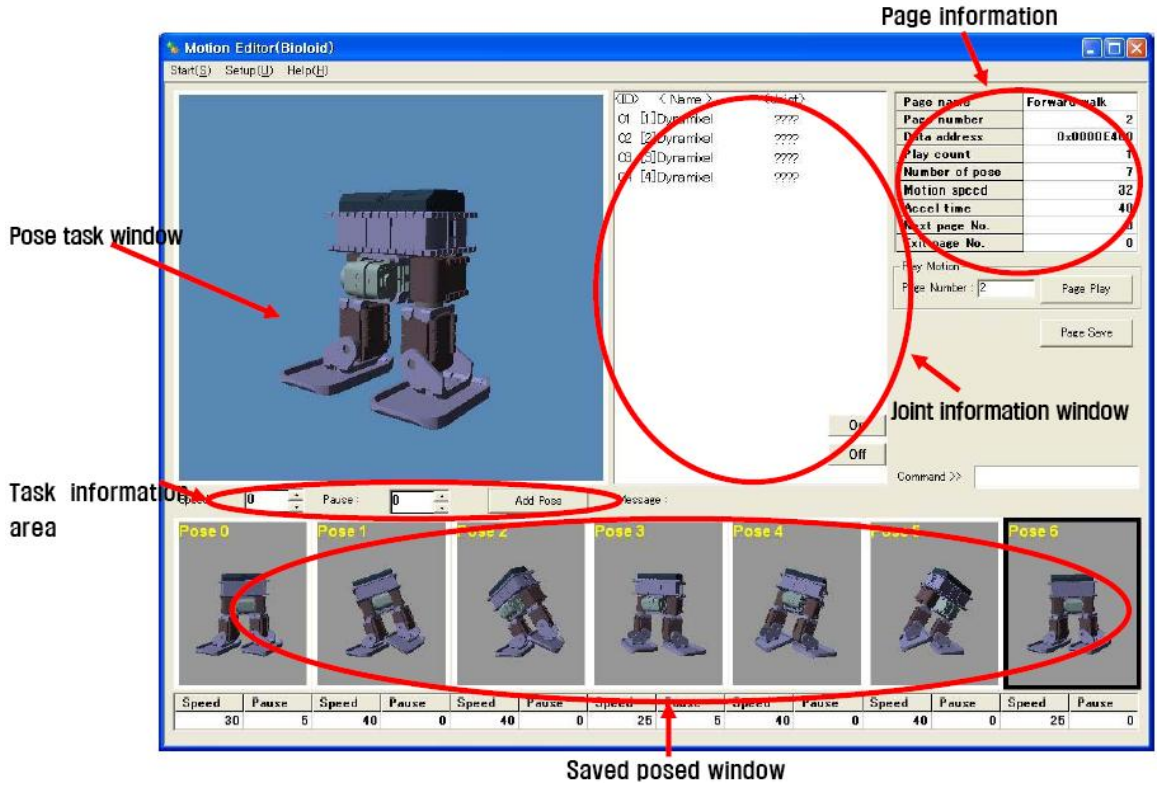


Figure 12 - Motion Editor Screen Description

The figure shows the Motion Editor screen showing the page information for a walking droid model robot.

- The Pose task window shows the current pose selected to be edited. This 3D image of the robot is synchronized with the actual one, implementing the pose on the robot as it is edited on the Motion Editor and vice versa. Once edited, they can be saved in the page by simply dragging the pose down to the Saved Pose Window.
- The Saved Pose window shows the poses in the current page being accessed. These poses can be individually accessed, edited and played on the robot.
- The Task Information Area shows the **Pose Speed** and the **Pose Pause** of the current pose selected.
  - The Pose Speed is the speed at which the pose is played on the robot. The default value of this parameter is 32. Higher value increases the speed such that a value of 64 doubles the pose speed.
  - The Pose Pause value illustrates a time delay to wait for after the pose finishes playing, before moving on to the next pose. The value specifies a delay of 7.8msec per this value.
- The joint Information window shows the current join positions of the robot. These numeric values can be increased or decreased by the user to create a certain pose. A torque on/off button is also available to turn the torque of the servos off and set the robot into a desired pose manually by moving each servo by hand. This allows more complex poses to be created easily.

- The Page Information window shows the information of the selected page.

Page name	Init
Page number	1
Data address	0x0000E200
Play count	1
Number of pose	1
Motion speed	32
Accel time	32
Next page No.	0
Exit page No.	0

  - **The Page Name** field is used to name each pose for convenience and to easily recognize the function of each page. The name is empty as default.
  - **Page number** is a unique number for each page saved in the CM-5 flash memory. The memory can store up to 128 pages at one time so the page number range from 0 – 127.
  - **Data address** shows the physical address of the page in the robot's memory.
  - **Play Count** depicts the number of times the page is supposed to be played. Increasing this value from 1 will make the robot loop through the page that number of times before moving on the next page or ending the motion.
  - **Number of poses** depicts the number of poses in the specific pose.
  - **Motion Speed** is the speed of the page's speed. This value allows the user to adjust the play speed of the whole page. The default value is 32 and, similar to the Pose Speed, doubling the value, will double the speed of the motion of the whole page. The overall motion speed is hence a product of both the Page Speed and the Pose Speed.
  - **Acceleration Time** shows the acceleration and deceleration time for each servo when it moves.
  - **The Next Page No.** points to the next page number to be executed when the current page is finished being played. This feature can be used to join pages together to create a complex motion, that single page cannot hold.
  - **Exit Page No.** points to the page to execute if an external signal indicates the robot to stop. The motion stops after executing this page to smoothly finish off the motion it is performing.
  - **Page Save Button** can be used to save the page into the memory.
  - **Page Play Button** plays the current page on the robot.

The Bioloid's memory can store up to 127 motion pages, with each page containing up to 8 blocks. The first block contains the page information while

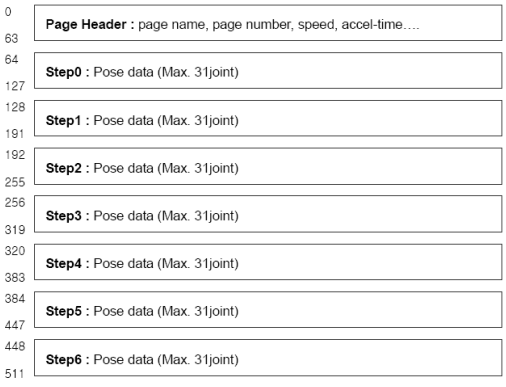


Figure 13 - Bioloid's flash memory division

the rest contain 7 poses (each being 64 bytes). The page size, thus, is 512 bytes. [14]

iii. Robot Terminal

The Robot Terminal software is a command based application provided for low-level management of the Bioloid robot. It connects the computer to the CM-5 through a serial link and can be used to transmit specific commands to the CM-5 module, monitor its status and setup configurations for the Dynamixels. The terminal is a very useful tool for debugging purposes and can be used to monitor the output of the CM-5 on the screen. The software can also be used to replace the default firmware and download files and load user developed programs onto the robot.

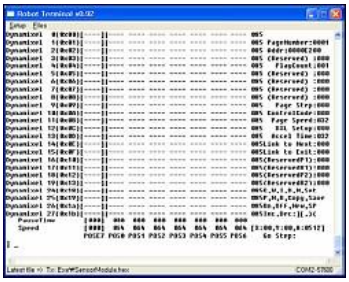


Figure 14 - Robot Terminal Screenshot [18]

The Robot Terminal can also be used to access and edit the Motion Pages in the flash memory, with simple commands, hence providing nearly all the functions of Motion Editor Software but in textual interface. [17] [18] [19]

2.2.3. Communications

i. Wiring

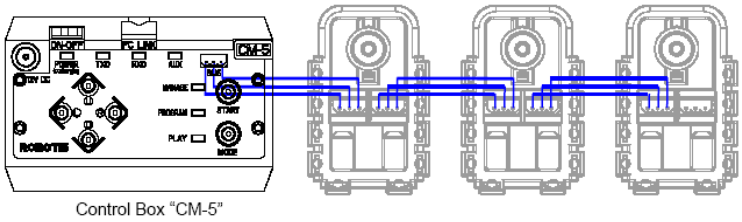


Figure 15 - Wiring of CM-5 Module with Dynamixels [9]

The Bioloid is based on a simple UART serial network, connecting the CM-5 module with AX-12 servos and the AX-S1 sensors using a single 3-wire serial bus, as shown in the figure above. Only a single line on the bus is used for both receiving and transmitting data. A 74HC126 quad buffer is used to multiplex the RxD (receive) and TxD (transmit) data onto the data line. This enables a Half-Duplex multi-dropped serial network configuration to be implemented with 3 wires. The main controller needs to convert its UART signals to half duplex type. The figure below shows a circuit diagram of the internal circuit of the CM-5 module which enables it to do the required operation.

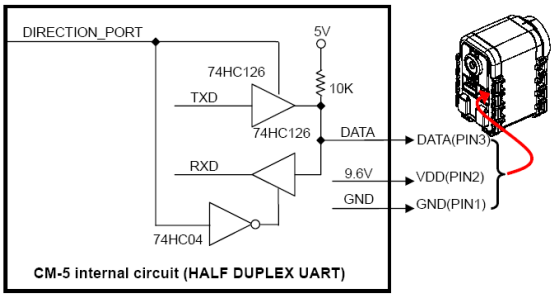


Figure 16 - CM-5 Internal Circuit Diagram [9]

The status of the DIRECTION_PORT determines the direction of the data signals. When the DIRECTION_PORT is High, the signal TxD is output as Data. Otherwise, the signal Data is input as RxD. [9] [17]

ii. Protocol

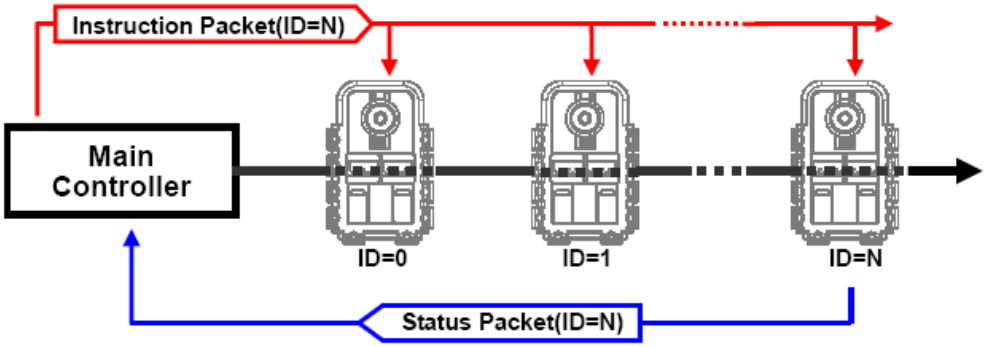


Figure 17 - Bioloid Communication Protocol

The CM-5 controller communicates with the Dynamixels through asynchronous serial communication by sending and receiving digital packets. The CM-5 uses the unique Network IDs to address the desired Dynamixel and sends an instruction packet with the set ID and an instruction. Only the Dynamixel with that specific ID will respond to the packet and send back a Status Packet, with the required parameters requested (as depicted in the figure above). [9]

The structure of the Instruction packet is shown in the figure below:

Instruction Packet	OXFF	0XFF	ID	LENGTH	INSTRUCTION	PARAMETER1	...PARAMETER N	CHECK SUM

- The two 0xFF bytes indicate the start of the packet.
- The ID contains the unique ID of the Dynamixel being addressed.
- The Length field carries the length of the packet. Its value is computed by the formula "Number of parameters + 2".
- The Instruction byte carries the required instruction to be carried out by the Dynamixel.
- Parameter carries the additional information needed to carry out the instruction. For example, if writing to a certain address of a servo, the address as well the data to be written is contained in this field of the packet.
- Finally a checksum sum is attached to assure successful transmission of the packet.

The instruction set of the AX-12 servo is given in the Appendix-D.
The status packet, shown in the figure below, has a structure very similar to the instruction packet.

Status Packet | 0XFF | 0XFF | ID | LENGTH | ERROR | PARAMETER1 | PARAMETER2 ... | PARAMETER N | CHECK SUM |

- The two 0xFF bytes indicate the start of the packet.
- The ID contains the unique ID of the Dynamixel sending the Status Packet.
- The Length field carries the length of the packet. Its value is computed by the formula "Number of parameters + 2".
- The Error byte contains a value indicating the type of error, if any. Each bit of the byte represents a specific error shown in the table below.

Table 2 - Status Packet Error Byte Structure [9]

Bit	Name	Details
Bit 7	0	-
Bit 6	Instruction Error	Set to 1 if an undefined instruction is sent or an action instruction is sent without a Reg_Write instruction.
Bit 5	0	
Bit 4	Checksum Error	Set to 1 if the checksum of the instruction packet is incorrect
Bit 3	Range Error	Set to 1 if the instruction sent is out of the defined range
Bit 2	Overheating Error	Set to 1 if the internal temperature of the Dynamixel unit is above the operating temperature range as defined in the control table.
Bit 1	Angle Limit Error	Set as 1 if the Goal Position is set outside of the range between CW Angle Limit and CCW Angle Limit.
Bit 0	Input Voltage Error	Set to 1 if the voltage is out of the operating voltage range as defined in the control table.

- Parameter carries the additional information needed to carry out the instruction. For example, if writing to a certain address of a servo, the address as well the data to be written is contained in this field of the packet.
- Finally a checksum sum is attached to assure successful transmission of the packet. [9] [14]

2.3 Chapter Summary

This chapter detailed the background knowledge of the project. It started with giving a brief introduction to the field of Robotics. It emphasized the importance of humanoid robots and their development history in the industry.

The chapter looks into the details of the architecture of the Dynamixel AX-12 servo, the AX-S1 Sensor and the CM-5 module of the Bioloid Kit. It described the software development kit provided with the Bioloid Kit; The Behaviour Control, the Motion Editor and the Robot terminal. It then, talked about the communication protocols of the CM-5 module and the Dynamixels.

The next chapter will detail the development and implementation of C programming to control the robot.

3. Description and Discussion of Work Done

This chapter gives a detailed explanation of the software developed in order to meet the objectives of the project. The chapter begins with a brief introduction to the Bioloid Humanoid model and looks in to its assembly.
It then gives a brief introduction to the compiler used to implement the software developed. Further on, it explains the software developed in C language to achieve the project aim. It gives a detailed explanation of the design and working of each function developed and states why they were important.
It then looks in to the steps taken to test the system developed and debug the problems that occurred.
It finally ends by presenting the results and outcomes of the system developed.

3.1. The Bioloid Humanoid Model

As stated earlier, the project represented by this report was based on the humanoid robot model built from the Bioloid Comprehensive Kit. The robot was built according to the instructions provided in the Assembly Manual Guide, provided with the Kit. [22]

The robot was built using 18 AX-12 servos, hence have 18 DOF. Each servo operates as a joint of the robot. The AX-S1 sensor is mounted on the top, as the head of the robot such that the IR sensors can sense from the front, left and right of the robot. The diagram below shows the configuration of the robot

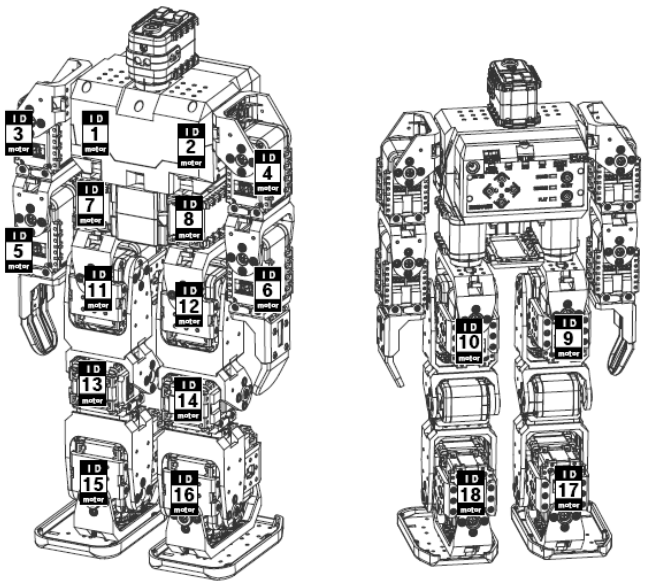


Figure 18 - Bioloid Humanoid Model [22]

The humanoid model was already built and provided by the University and the project work was then implemented on it.

3.2. WinAVR Compiler

As the focus of the project presented was to program the Bioloid using C programming language, comprehensive research had to be done in order to understand the technique. Although programming the robot using C language is much more complex than simply using the software development kits provided, it allows complete control over the robot. In order to program the controller using C language, an appropriate C compiler had to be used.

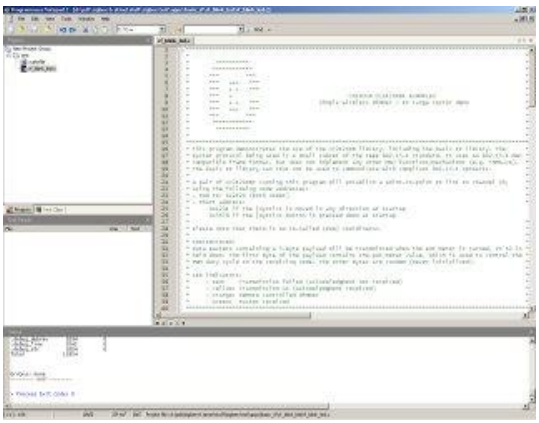


Figure 19 - WinAVR Comiler Screenshot [17]

Since the CM-5 module is based around the Atmel Atmega128 microprocessor, the best option for a programming environment was discovered to be the **WinAVR** package available by AVRFreaks. Although many different compilers can be used to write and compile programs to the ATmega128 microprocessor, the WinAVR environment was chosen for its various advantages. The biggest advantage of it is that it is freeware and can be downloaded and installed on as many computers as desired. WinAVR consists of a suite of executable, open source software development tools for the Atmel microprocessors. It includes a GCC compiler coupled with a freeware editor, **Programmers Notepad**, which allows programmers to write, edit and debug codes. Not only that, it comes with a free utility, **mfile**, which can be used to generate the project makefiles. Makefiles specify how to derive a target program from a number of sources it is depended upon. The WinAVR environment also includes, **AVRLib** a collection of function libraries which can be used by simply calling them and hence will not have to be written from scratch. [17] [20]

Appendix – F gives a detailed guide of how to use the compile the code and download it on the robot's main controller.

3.3. Design and Implementation

In order to accomplish the aim, a C program was developed. A modular approach was adapted to complete the program and hence it was divided into a number of smaller functions. The table below lists these functions stating their input and output arguments and gives a brief description of each. Each function is explained in detail later on in the chapter. The Appendix E contains the relevant source codes explained.

Table 3 - List of Functions

Function Name	Input	Output	Description
PortInitialize	None	None	Initialises the ports of the CM-5's microprocessor
SerialInitialize	Port, Baud Rate, Interrupt	None	Initialises the Serial port connection
TxPacket	ID, Instruction, Parameter Length	Packet Length	Sends the instruction specified to a specified servo
RxPacket	Packet Length	Packet Length	The function reads the buffer for a status packet for the packet length specified.
TxD8Hex	Data	None	Displays the hexadecimal value on the screen.
TxDString	Data	None	Prints the Data on the screen as a string.
getCurrentServoPos	ServoNo	Current Position	Reads the current position of a specified servo
SendServoTargetPos	ServoNo, Target Position, Speed	None	Sends a specified servo to the target speed passed at the Speed value passed.
ReadPose	Page No., Pose No.	Pose	Reads the motion pose saved on the flash memory of the CM-5 module
MaxDistance	Pose	Max	Calculates the maximum distance a servo has to move amongst all the servos to play a pose.
doPose	Page No., Pose No.	Pose Pause	Sends the robot to specified pose of a page.
doPage	Page No.	Page Pause	Plays the motion page on the robot.
doMove	Page No.	None	Plays motions pages in a sequence on the robot.
CheckLeftSensor	None	Flag	Reads the value of the left sensor and returns a Boolean flag depending on the value
CheckCenterSensor	None	Flag	Reads the value of the front sensor and returns a Boolean flag depending on the value
CheckRightSensor	None	Flag	Reads the value of the Right sensor and returns a Boolean flag depending on the value

3.3.1. Example code functions

In order to accomplish the aim, the first objective was to implement functions to send instruction packets to the Dynamixels and receive status packets from them. Fortunately, the User Guide to the AX-12 servo provided an example C code which included the basic '#define' directives defining the AX-12 servo's control table addresses. It also included functions to configure the serial communication between the CM-5 microcontroller and the Dynamixels. Also, the code included functions to transmit instruction packets and receive status packet from the Dynamixels. A few other functions to print the from the CM-5 module to the Robot Terminal screen were also included. These functions were acquired from the

example code and used for the project. The functions taken from the example code are in the 'Bioloid.c' files of the project, attached in the Appendix E.

The operation of the functions used is briefly explained below:

- **void PortInitialize(void)**
  This function initialises the ports of the microprocessor to implement half duplex communication.

- **void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt)**
  This function initialises the serial port, the function is called for, to the initial state.

- **byte TxPacket(byte bID, byte bInstruction, byte bParameterLength)**
  This is function assembles the arguments passed to it in the format of the instruction packet (discussed in chapter 2.2.3) and sends it to the Dynamixel with the network ID equal to the bID argument passed.
  The function assembles these values in the specific format in a global buffer; gbpTxBuffer. The value 0xFF is stored in the first two locations of the buffer indicating the start of the Instruction Packet. The unique network ID of the Dynamixel is stored in the third location followed by the length of the packet in the fourth location. If there are any additional parameters to be sent, the length of these parameters are passed in the function as bParameterLength, and this value is added to the packet length. The instruction to be sent, bInstruction, is stored in the fifth location of the buffer. Finally, the checksum is calculated and stored in the end of the buffer. The value of the direction port is then inverted to send the instruction to the specific Dynamixel.
  The function returns the length of the packet calculated.

- **byte RxPacket(byte bRxPacketLength)**
  This function reads the status packet returned from a Dynamixel after it has been sent an instruction packet. The function implements a simple 'while' loop and reads the data off a buffer, gbpRxInterruptBuffer and stored it in gpbRxBuffer. Once the data is read, the function verifies the packet checking if any errors have occurred in the transmission.

- **void TxD8Hex(byte bSentData)**
  This function prints the data passed to it on the Robot Terminal screen as separate hexadecimal numbers.

- **void TxDString(byte *bData)**
  The function prints the data passed to it on the Robot Terminal screen as a string of characters.

3.3.2. Servo Functions

To accomplish the aim of the project, the first objective to overcome was to implement C language functions to get the current position of the AX-12 servo and to move it to a target position. These functions are displayed in detail below:

- **Reading a Servo's Position**

  The `'getCurrentServoPos(ServoNo)'` function reads the current position of a servo. The function loads the address of the location where the current position (`P_PRESENT_POSITION_L = 0x37`) is saved in the servo, and the number of bytes to read, in this case '2' (lower and higher byte of the current position), into the `gbpParameter` buffer. The function then sends a READ instruction to the servo using the earlier discussed `TxPacket()` function. The servo in return saves the current position of the servo in the `gbpRxBuffer`, which are read off it and returned by this function.

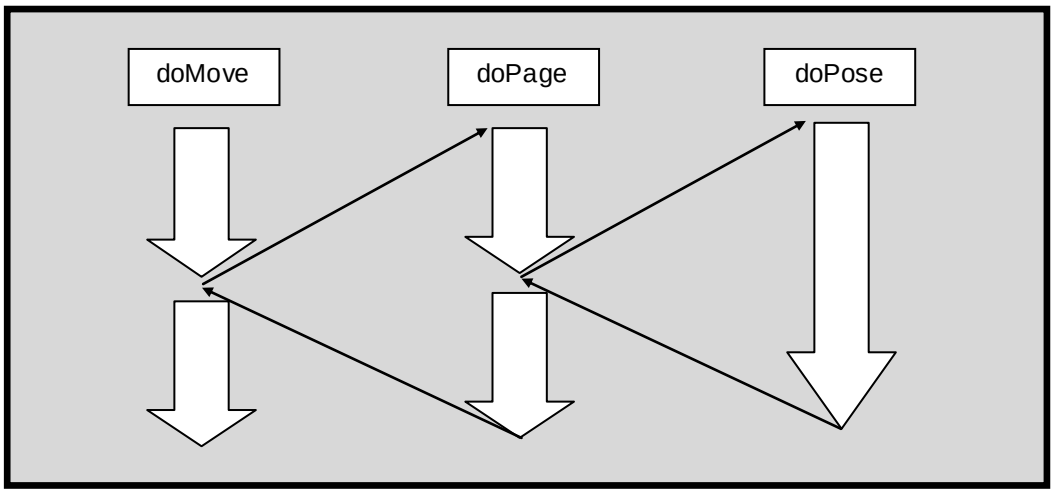
- **Sending a Servo to a Target Position**

  The function `SendServoTargetPos (ServoNo, Target, Speed)` written sends the servo to a target position at the specified speed. The function is passed in the servo's unique network ID, the target position and the speed. The function loads the address (`P_GOAL_POSITION_L = 0x32`) where to write these specific values these values and the parameters passed into the `gbpParameter` buffer. The function then sends the servo a WRITE instruction using the `TxPacket()` function. Once the values are written in the specific location, the servo's microcontroller automatically moves the motor to the target position.

The complete code for the Servo functions, explained above, is attached in Appendix E and is contained in the 'ME.c' source file.

3.3.3. Motion Editor Functions

Once the functions to operate the servos were completed, the next objective was to implement function to read and play the motion editor pages. The motion created by the Motion Editor plays in a sequence of motion Pages, which in turn is played by a sequence of motion Poses.



In order to implement this sequence, three separate function; doPose(), doPage() and doMove() were written.

The motion poses and pages were created using the Motion Editor software. These pages were downloaded on the flash memory of the CM-5 module using the Behaviour Control program. To make the robot play the pages, this data was to be read from the flash memory. Before this data was read, the relevant information of the poses was grouped together as a single structure and was defined as 'MOTION_EDITOR_DATA' in the Bioloid.h. A global variable of this structure type was defined with the name of 'ME'. This was used to handle the sequence of the pages to be played.

```
        typedef struct
{
      uint16_t CurrentPage;
      uint16_t TotalTime;
      uint16_t PageSpeed;
      uint16_t MoveFinishTime;
      uint16_t NoOfPoses;
      uint16_t CurrentPose;

} MOTION_EDITOR_DATA;
```

The information contained by a pose was similarly grouped in a structure 'POSE' struct shown below. The ServoPos stored the current positions of all the servos of the robot and the ServoTarget stored the target positions the servo had to move to in order to implement the motion pose.

```
typedef struct
{
       uint8_t PoseSpeed,ServoNo;
       uint32_t PosePause;
       uint32_t PoseAdr;
       uint16_t ServoPos[NUM_OF_SERVOS_ATTACHED];
       uint16_t ServoTarget[NUM_OF_SERVOS_ATTACHED];
} POSE;
```

The data was then read and stored in the relevant variables using the function described below:

- **Reading the Motion Pose Data**

This function `'ReadPose(Page, Pose)'` was written to read the motion data from the flash memory of the CM-5 and save it in the appropriate variables of the structures discussed above.
The base address of the Motion Editor data in the flash memory is 0xE000. Each motion page, of size 512 bytes, contained 7 poses, each of 64 bytes each. The first 64 bytes of the page contained page settings.

Each pose contains Pose Pause and Pose Speed, followed by the Target positions of each servo of the robot.

Table 4 - Offsets of page settings within a Page

Address	Description
0x00010	Page Fast Flag
0x0000F	Page Play Count
0x00014	Number of Poses in Page
0x00016	Page Speed
0x00018	Page Acceleration
0x00019	Next Page Number
0x0001A	Exit Page Number

Table 5 - Offsets of pose settings within a Pose

Address	Description
0x0003E	Pose Pause
0x0003F	Pose Speed

The function first declares a variable `Read_Pose` of type POSE. The pose read is stored in this variable. The function first reads the current positions of the servos connected in a loop and saves them in the '`Read_Pose.ServoPos`' array. The function then uses a 'pgm_read_byte_far' to read the data off the flash memory, using the calculated addresses, after disabling the interrupts of the system. Interrupts are disabled to disable them from interrupting the system while memory is being accessed. They are enabled back on after reading the memory.

$$Pose\ Pause\ Address = \{Pose\ Base\ Address + Pose\ Offset + (Page\ Size\ x\ Page\ Number) + (Pose\ Size\ x\ Pose\ Number) + Pose\ Pause\ Offset\}$$

$$Pose\ Speed\ Address = \{Pose\ Base\ Address + Pose\ Offset + (Page\ Size\ x\ Page\ Number) + (Pose\ Size\ x\ Pose\ Number) + Pose\ Speed\ Offset\}$$

The function also reads the Page Speed of the Page and stores it into the global variable ME created (discussed earlier in the chapter).

A Page Fast value is also read which determines if this page should be played at a higher speed than specified. This value is not visible in the Motion Editor software. If this flag value is set, the function sets the Page Speed to a value of '160'.

Once the pose settings are read, the function moves on to read the target positions of each servo to be moved to. The values are read in a loop and saved in the `Read_Pose.ServoTarget` array.

The function in the end returns the `Read_Pose`.

- **Playing a Pose**

To make the robot play a motion pose, all the servos of the robot had to be moved to the target positions specified in the pose simultaneously. Also, a strategy had to be implemented in order to make all the servos reach their target positions at the same time, hence playing a pose. This was done by figuring out which servo of all had to move the maximum distance. That specific servo would then move at the maximum allowed speed, while all the other servos would move proportionally slower at a calculated speed depending on the distance they have to move in respect to the maximum distance.

The maximum distance was figured out by creating a simple function, `MaxDistance(Read_Pose)`, that searches through all the servos and calculated the difference of each servos target position, defined in the motion pose passed to it, and the current position of the servo. The function then returned the maximum moved by a servo.

Once the maximum distance was known, a function, `doPose(page, pose)`, was written which calculates the appropriate speed for each individual servo and makes them move to their respected target positions at that speed. The function takes as arguements the page number and the pose number to play. It uses the `ReadPose(Page, Pose)` function to read the specified pose information. It then calls the `MaxDistance(Read_Pose)` function to calculate the maximum distance and moves on to calculate the speeds for the servos.

The motion playback is influenced by two values; Pose Speed and Page Speed. It was found that if both these values are set to their default value (32), the speed of the servos would be 90˚/sec. So to make things simpler, a normalised speed (`NSpeed`) was calculated which took in account the product of both these speeds divided by their defaults values. Thus, if the Page Speed and the Pose Speed of a motion page are both equal to their default values, the value of `NSpeed` would be 1.

$$Normalised\ Speed = \frac{Page\ Speed}{32} \times \frac{Pose\ Speed}{32}$$

The maximum allowed speed was calculated using this normalised speed. The speed of the AX-12 servo ranges from 0 – 1023 units. According to the AX-12 Servo's Manual, the maximum value of 1023 makes it move at 114 revolutions per minute. To calculate the servo speed, the conversion rate from speed in deg/sec to servo speed units was calculated:

$$\begin{aligned} 1023\ servo\ units = 114\ rpm &= \frac{114}{60} \\ &= 1.9\ rps \\ &= 1.9 \times 360 \\ &= 684^{\circ}/sec \end{aligned}$$

Since, if the NSpeed = 1, the servo moves at 90˚/sec, the conversion factor from NSpeed to Servo Speed units is:

$$\frac{90}{684} \times 1023 = 135$$

Thus, the motor moving at maximum speed was made to move at MaxServoSpeed = NSpeed × 135. While the other servo's speeds was programmed to move at a fraction of the `MaxServoSpeed`, depending on how far it has to move comparing to the servo that has to move the maximum distance.

$$Servo\ Reference\ Speed = Maximum\ Servo\ Speed \times \frac{(Distance\ a\ servo\ has\ to\ move)}{Maximum\ Distance}$$

A simple loop was implemented to send the servos to their respective target positions at the appropriate speed calculated for each of them.

```
/*Calculate the speed of each servo in reference to the Servo Moving at
the Max speed. (one that has to move the max distance.) */
        ServoRefSpeed = (uint16_t)((float) MaxServoSpeed *
        ((float)Difference/(float)Max));

        if (ServoRefSpeed>MaxServoSpeed)ServoRefSpeed = MaxServoSpeed;

        SendServoTargetPos(ServoNo,Read_Pose.ServoTarget[ServoNo],ServoRefS
        peed);

        Read_Pose.ServoPos[ServoNo] = Read_Pose.ServoTarget[ServoNo];
        }
        _delay_ms(ME.MoveFinishTime + 10);
```

Once the command to send the servos was issued, the program had to wait for the servos to reach there target positions. Otherwise, if a new pose is played right after one pose, the robot would start playing the new one when its half way through the first one. This problem was faced during the function was being implemented. A strategy to keep checking if all the servos have reached there target positions and waiting until then, was first implemented. But the reading the positions of each servo repeatedly added a delay, making the motion unsteady. Thus, to overcome the issue, a delay was added to the function waiting till all the servos have reached their respective target positions. The Bioloid User's Guide provides the formula to calculate the time it takes for a pose to finish playing:

$$Move\ Finish\ Time = 7.8msec \times Time \times \frac{32}{PageSpeed}$$

$$Since\ Time = \frac{Distance}{Speed} = \frac{Max\ Distance}{3.41} / (Normalise\ Speed \times 90)$$

[Where 3.41 is servo distance units = 1˚]

The function returns the value of the time to pause after the pose has been performed. This time is calculated by the formula given in the User's Guide:

$$Pose\ Pause\ (msecs) = 7.8ms \times Pose\ Pause\ Value \times \frac{32}{Page\ Speed}$$

- **Playing a Page**

Once the function to play a pose was done, a function name `doPage(page)` was implemented. The function takes the Page Number as an argument and simply plays the poses in the page. The function first reads the total number of poses in the page and then runs a loop till that value. The function calls the `doPose(page, pose)` function to play each pose in the page in a sequence. The function implements the pose pause by calling a delay function which waits for the Pose Pause time returned by the `doPose(page, pose)` function.

```
uint16_t doPage(int page)
{
        if(page != -1) {
                ME.CurrentPage = page;
                cli();                  //disable interrupts
                ME.NoOfPoses= pgm_read_byte_far
                ((uint32_t)POSE_BASE_ADR +
                (uint32_t)PAGE_SIZE *
                (uint32_t)ME.CurrentPage +
                (uint32_t)PAGE_NUM_OF_POSES);
                sei();                  //enable interrupts

                ME.TotalTime = 0;
                ME.CurrentPose = 0;

                do {
                        _delay_ms(doPose(ME.CurrentPage,
                        ME.CurrentPose));

                        ME.CurrentPose++;

                } while (ME.CurrentPose < ME.NoOfPoses);
                return ME.TotalTime;
        }
        return 9999;
}
```

- **Playing a Motion**

  Most of the simple motions were implemented by simply playing a single page. But complex motions like walking and turning required playing more than one pages in a sequence. These motion pages were linked together by the Next Page value in the page settings.

```
void doMove(int page)
{
        uint8_t NextPage = page;
        do
        {
                ME.CurrentPage = NextPage;
                cli();  //disable interrupts
                NextPage= pgm_read_byte_far
                ((uint32_t)POSE_BASE_ADR +
                (uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage
                + (uint32_t)PAGE_NEXT_PAGE);
                sei();  //enable interrupts

                _delay_ms(doPage(ME.CurrentPage));
        } while (NextPage!=0);
}
```

  A function, `doMove(page)`, was created which plays the page passed to it and then moves on to play the page pointed by the Next Page of that page. The function keeps on playing the pages, until the Next Page value is equal to zero. The function also implements a delay equal to the Page Pause value of the page. Eventually playing a motion designed using the Motion Editor software.

The complete code for the Motion Editor functions, explained above, is attached in Appendix E and is contained in the 'ME.c' source file.

3.3.4. Sensor Functions

Keeping the aim of the project in focus, the next objective to tackle was implementing functions to operate the AX-S1 Sensor module. The humanoid had three pair of sensors; one on each side – left, right and front. Each sensor senses automatically senses the distance from the object in front of it and stores a value ranging from 0-1023. Higher the value, closer the object is. In order to read this value, the address of the location of this value was defined in a header file [*Sensor.h*].

Initially, a function was implemented to read the Obstacle Detection Flag of the sensor module (discussed in Chapter 2.2.1) but the flag is only set comparing to a single compare value for all three sensors. This was overcome by implementing three separate functions for each side's sensor which read the value of the sensor, compared it with a set threshold and returned 1 if the value was greater than the threshold, indicating that an object is close to that side of the robot.

`CheckLeftSensor()` function sends a READ instruction to the sensor module, specifying the address of the left Infrared sensor data. After verifying the returned values status and if no error had occurred in the transmission, the function compares the value with a preset threshold value for the sensor and returns '1' if the value is greater than the threshold depicting that there is an object on the left side of the robot. A '0' is returned if the value is smaller than the threshold or an constant error value (`SERVO_NOT_CONNECTED`) is returned in case of an error.

```
uint8_t CheckLeftSensor()
{
        uint8_t LeftObstacle;

        gbpParameter[0] = S_LEFT_IR_SENSOR_DATA;
        gbpParameter[1] = 1; //Read Compare Bit

        TxPacket(SensorID,INST_READ,2);
        uint8_t bRxPacketLength =
RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);

        if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE +
gbpParameter[1])
        {
                if(gbpRxBuffer[4]==0)  {
                        LeftObstacle = gbpRxBuffer[5];}
                else{
                        // error
                        return SERVO_NOT_CONNECTED;}
        }
        else{
                //error
                return SERVO_NOT_CONNECTED;}

        if (LeftObstacle > LEFT_OBSTACLE_THRESHOLD)
                return 1;
        else
                return 0;
}
```

Similarly, `CheckRightSensor()` function checks the right sensor and `CheckCenterSensor()` function checks the front sensor.

Appropriate values for the thresholds for each side's sensors were chosen through the 'hit and trial' method.

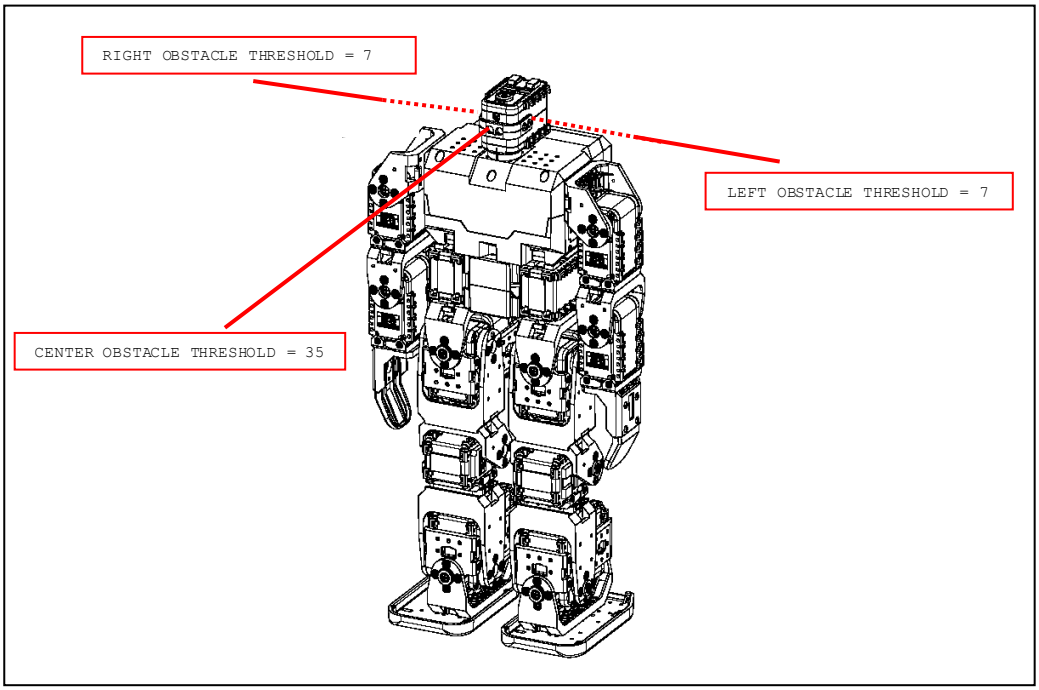


Figure 20 - Bioloid Humanoid Sensor Directions

The complete code for the Sensor functions, explained above, is attached in Appendix E and is contained in the 'Sensor.c' source file.

3.3.5. Maze Solving

Once the functions to play a motion as well as the functions able to sense distance were implemented, the next task was to implement a simple maze solving technique which would enable the robot to find its way out of a simple maze. Although there are number of automated maze solving methods, the technique chosen for the project was the wall-following technique.

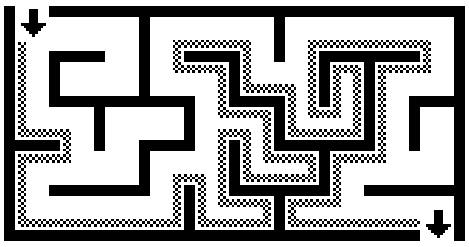


Figure 21 - Right-hand wall following algorithm example[21]

Wall following is a simple, yet effective technique to solve a simple maze. The strategy is to keep contact with on of the wall of the maze, either left or right, and follow it and until the exit is found. The technique works only if all of the walls of the maze are connected together. Since the project was based on solving a simple loop, this

Figure 22 - Right-hand wall following example [21]

technique was chosen to be the best option. The figure on the right shows an example of a traversal of a simple maze using the right-hand wall following technique. [21]

Since the functions to play motion pages had already been implemented, they could be used to make the robot walk or turn in either direction using the motion pages created from the Motion Editor software. The wall following algorithm was thus implemented based on the right-hand rule, by writing a function which calls the `doMove()` function to make the robot take the appropriate route depending on the input from the sensors. The flow chart below shows the operation of the function:

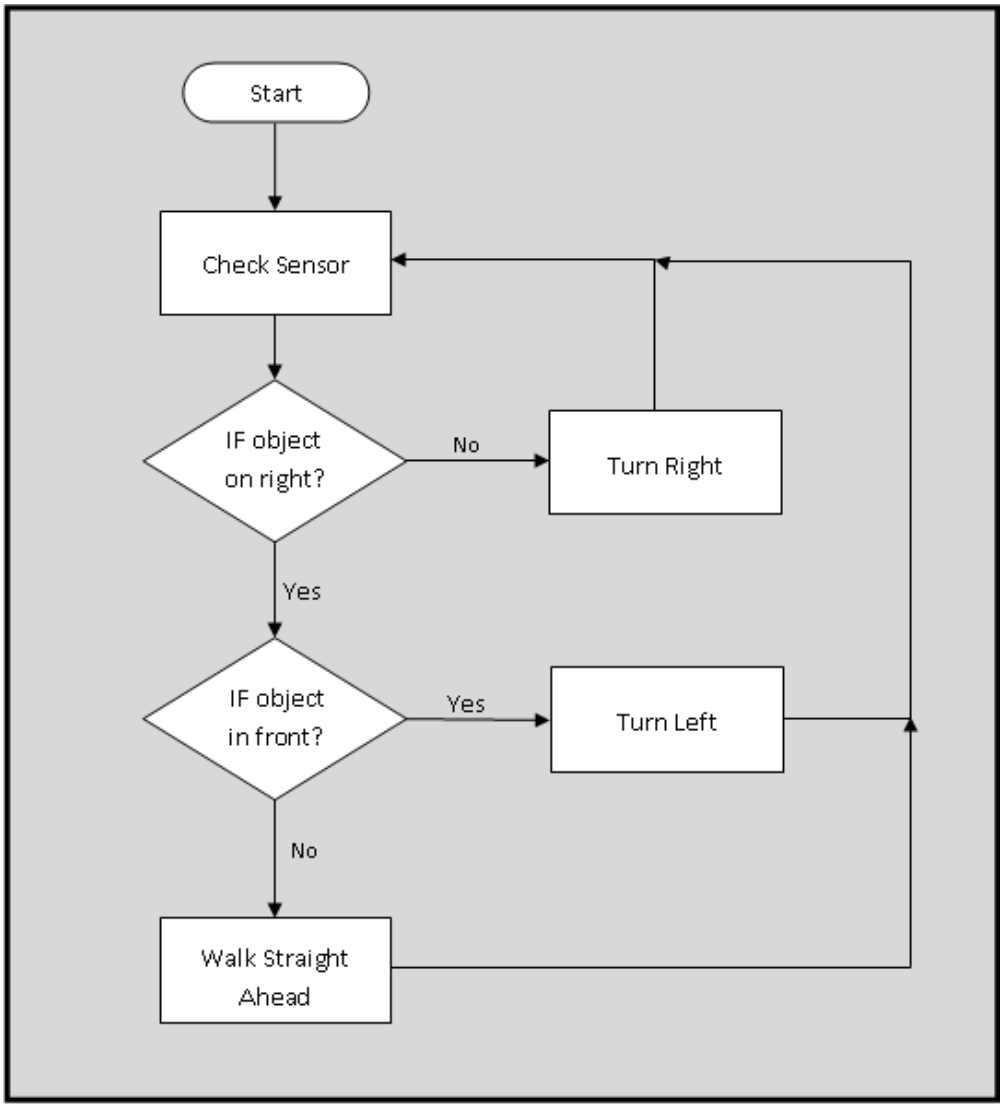


Figure 23 - Flowchart representing the 'decision making' wall following algorithm developed.

The complete code for the Wall Following function, explained above, is attached in Appendix E and is contained in the 'Main.c' source file.

3.4. Testing and Debugging

Once the all the required functions to achieve the objectives of the project were implemented, they were carefully tested and debugged to achieve optimum results. Initially, each function was separately tested and the result was analysed. Although most of the functions produced pleasing results, some had to be adjusted to achieve the best results.

One of the major problems faced was making the motion pages for walking and turning stable. The demo pages provided with the kit were very unstable and the robot kept falling backwards when played. This was overcome by configuring the offset settings of the pages through the Motion Editor software. Each pose of the walking and turning pages were individually tuned to make the motion more stable. Although the task was tedious, the results were worth the effort. The walking pages were satisfying, the turning pages, however, still need some adjustments and can be made much more stable.

Also, when the walking motion was initiated from a standing position of the robot, the robot's inertia would make the robot fall. This was overcome by modifying the `ReadPose()` function and decreasing the speed of the first page of each motion. This made the motion start smoothly and in the final run, the robot walked efficiently.

Another problem faced was caused by the timing issue. The delay function created was not working properly, since the compiler would automatically skip through the loop implemented in the function. Thus, a built-in function from the AVRLib library was acquired. But even with the new delay function, the desired delay could not be produced. The User Guide mentions that each unit of Pose Pause and Page Pause results in a delay of 7.8m seconds. Implementing the same concept, a delay of the same amount of time was given but resulted in an unstable playback of the motion pages. Through experiment it was discovered that by adding an extra delay of 10m sec, the best results were achieved. Thus, a modification was made in the `doPose()` function and the extra delay time was added.

With these problems resolved and some other minor adjustments made, all the functions worked fined. These functions were finally put together and tested by running the wall following program designed.

3.5. Outcomes

Once the testing of the program was complete, the final version of the program was run. To accomplish the aim of the project, a simple of the shape 'UH' (representing the University of Hertfordshire) maze was made out of cardboard and the robot, with the wall following program loaded in to it, was left to solve it. The robot, implementing the right-hand rule, followed the wall on its right and successfully solved the traversed the maze, finding its way out to the exit.

The maze designed is shown in the figure below. The arrowed-line shows the path taken by the robot while solving the maze.

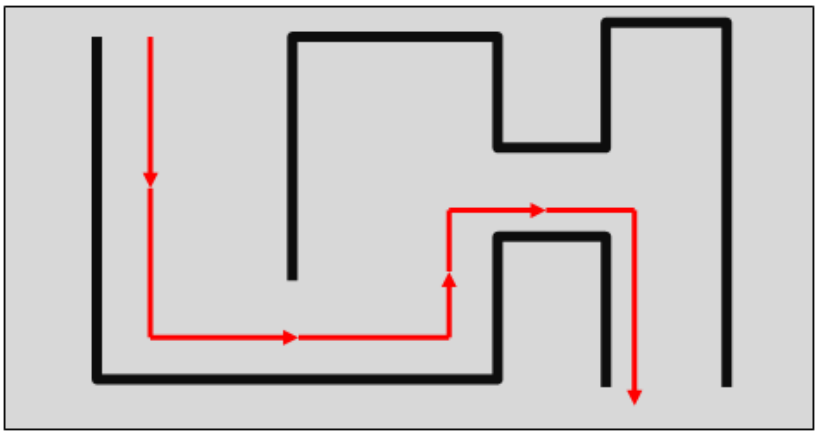


Figure 24 – Maze map and robot's route

The pictures shown below show the maze made up of simple cardboard boxes and the robot solving it.

Figure 25 - Robot solving the maze

3.6. Chapter Summary

This chapter gave a detailed explanation of the software developed in order to meet the objectives of the project. After giving a brief introduction to the Bioloid Humanoid model and how it was assembled, it introduced the WinAVR compiler used for writing the C language code. It explained why this compiler was chosen for the project.
It then moved on to detail the C functions implemented throughout the project describing the work done. It explained the design and operation of each function.
The chapter then described the testing process by looking into the limitations of the system, the problems faced and the actions taken to counter them.
It then presented the results and the outcomes of the software developed. The next chapter will present a summary of the project and review the status of the objectives.

4. Conclusion and Other Aspects

This chapter concludes the project work and gives a summary of the report. It looks in to the objectives laid down at the start of the project and the status to which each of those objectives were met.
It then looks into the future development possibilities of the project and gives examples of how it could be further improved.

4.1. Conclusion

In conclusion, the project completed a great deal. The project achieved the main goal to program the Bioloid humanoid robot in C language to make it solve a simple maze. The set objectives were met successfully and performed the required tasks. Initially, a comprehensive research was done about the background of the topic and the design and operation of the hardware and software used for the project. A design layout was then laid which was accomplished by writing C functions to operate the Dynamixel servos and sensor by sending simple instructions through the main controller module. Further on, functions were implemented which were used to play the motion sequences, created through the Motion Editor software, on the robot. Finally, a maze solving algorithm was coded which adapted the wall-following strategy to enable the robot to find its way out of a simple maze created.
Besides the minor issues of the instability of the robot's movements, the overall performance of the project was satisfying.

4.2. Review of Objectives

This section will review the project objectives highlighting their finished states and performances:

i. **Research and study the hardware components, existing software development kits and the communication protocols.**
This broadened the understanding of the operation, design, architecture and control of the servos, sensors and the microprocessor used to build the robot. The knowledge gained in this stage was used throughout the project and helped in implementing the project.

ii. **Devise functions in C programming language for the main controller module of the robot, to send simple commands to servos and read and write information from the servos and the sensor.**
The `getServoPosition()` function was implemented which allowed the CM-5 module to send a READ instruction to the servo, reading the current position of the servo. The function allowed the ability to monitor the positions of each servo.

The `SendServoTarget()` function was implemented which allowed to CM-5 module to send a WRITE instruction to the servo in order to send the servo to a target position. The function was a basic part of the project allowing the ability to move the servo as required.

iii. **Implement functions in C programming language for the main controller module of the robot, to read the motion data stored in the flash memory and play the motion sequences on the robot.**
This allowed the ability to read the motion pages developed using the Motion Editor Software, and play those motion sequences on to the robot. This allowed the ability to move the robot as desired making it walk, turn and stand.

iv. **Implement a simple wall following algorithm to make the robot find its way out of a simple maze.**
Finally, the last objective was met which made the robot successfully traverse a simple maze and find its way out to the exit.

4.3. Time Management

The time management plan made at the start of the project (attached in Appendix A) was completely followed although the initial research done was completed before the planned time and the report writing took more time than planned. Overall, all deadlines were met and the original plan had not to be changed much.

4.4. Further Development

The project lays down the basics of programming the CM-5 module using C programming and could be further developed and enhanced in many aspects.

Since the motion sequences developed for the walking and the turning were not very stable, further work could be done on them to make them stable enough for the robot to move without falling. A gyroscopic sensor module could be installed which would measure the stability of the robot and a program could be designed to make the robot automatically adjust its stance in order to prevent itself from falling. A pressure sensor could be installed on the foot of the robot for it to be able to detect the different textures of the surface it is walking on and enable it to adjust its walking patterns appropriately.

Also, a camera module could be installed instead of the AX-S1 sensor making it able to recognize different colours and detect objects. This will allow wide number of possibilities to be implemented.

A wireless controller, such as the RC-100 and the ZIG-100, could also be incorporated for wireless control of the robot. This would enable the robot's motion to be controlled by sending wireless commands through the controller.

There is also an option to install more servos to the humanoid model giving it more degree of freedom. For example, adding a servo at the waist of the model would allow the robot to turn its torso, allowing a whole new range of motion sequences to be implemented.

4.5. Chapter Summary

This chapter concluded the report by giving a brief summary of the project. It looked into the objectives that were set in the beginning of the project and explained to what extent each of those has been completed.

It discusses the time management required during the project discussing the time plan designed at the beginning of the project.
It then looked into the future development possibilities of the project and gives examples of how it could be further improved.

REFERENCES

[1]. Robot-Savvy. (2009) – '*Bioloid Comprehensive Kit Information'.*
http://robosavvy.com/store/product_info.php/products_id/82,
[Date accessed 08/04/09]

[2]. Jea-Kweon Han, In-Yong Ha and Byoung-Soo Kim. (2008) - '*Educational Robotic Construction Kit: Bioloid'*,
http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2008/data/papers/1723.pdf,
[Date accessed 08/04/09].

[3]. Wolf, Joerg. (2008), - *'The secrets of Robot Football'.*
http://www.swrtec.de/swrtec/research/publications/Secrets_of_RobotFootball.pdf
[Date accessed 08/04/09].

[4]. Azonano Technology (2007) *'US Navy Academy to Acquire 50 Robotis Bioloid Humanoid Robot Kits from KumoTek'*,
http://www.azonano.com/News.asp?NewsID=4498
[Date accessed 08/04/09].

[5]. Nordin, P. and Nordahl M. (date unknown) - *'An Evolutionary Architecture for Humanoid Robot'*
http://humanoid.fy.chalmers.se/text/cuba.pdf
[Date accessed 08/04/09].

[6]. Wikipedia (2009) – *'Humanoid Robot'*
http://en.wikipedia.org/wiki/Humanoid_robot
[Date accessed 10/04/09]

[7]. Kwoon, Woong. , Kim Hyun K., Park Joong K. , Roh Chang H., Lee Jawoo., Park, Jaeho., Kim, Won-Kuk. and Roh, Kyungshik – *'Biped Humanoid Robot Mahru III'*
http://planning.cs.cmu.edu/humanoids07/p/18.pdf
[Date accessed 12/04/09]

[8]. Biped Robots (2007) – *'Biped Robots'*
http://www.bipedrobots.co.uk/
[Date accessed 12/04/09]

[9]. ROBOTIS (2006) – *'Dynamixel AX-12 User's Manual'*

[10]. Moballegh, Hamid R., Guilbourd, Roman., Hohl, Gretta., Oertel, Michael. and Rojas, Raul. (2007) – *'FUmanoid Team Description 2007'*
http://mail.fibo.kmutt.ac.th/robocup2008/store/FUmanoid_Germany_TDP_01.pdf.pdf
[Date accessed 12/04/09]

[11]. Simpson, Michael. (2007) – *'Understanding the AX-12'*
http://www.antratek.nl/pdf/Understanding-AX12.pdf
[Date accessed 13/04/09]

[12]. ROBOTIS (2006) – *'Dynamixel AX-S1 User's Manual'*

[13]. Robot Italy (2004) – *'AX-S1 Sensor Specifications'*
http://www.robot-italy.com/product_info.php?products_id=733
[Date accessed 14/04/09]

[14]. ROBOTIS (2006) – *'Bioloid User's Manual'*

[15]. Roboporium (2004) – *'Bioloid Comprehensive Kit'*
http://www.roboporium.com/bioloid_comprehensive.html
[Date accessed 14/04/09]

[16]. Trossen Robotics (Date unknown) – *'Bioloid CM-5 Control Module'*
http://www.trossenrobotics.com/bioloid-cm-5-control-module.aspx
[Date accessed 14/04/09]

[17]. Robot-Savvy. (2009) – *'Bioloid Comprehensive Kit Knowledgebase'*
http://robosavvy.com/site/index.php?option=com_content&task=view&id=82&Itemid=81
[Date accessed 15/04/09]

[18]. Crust Crawler Robotics (2007) – *'Bioloid Robotics Kit'*
http://www.crustcrawler.com/products/bioloid/index.php?prod=59
[Date accessed 15/04/09]

[19]. Trossen Robotics (Date unknown) – *'Bioloid Robot System'*
http://www.trossenrobotics.com/bioloid-robot-system.aspx
[Date accessed 16/04/09]

[20]. San Jose State University (2008) – *'Introduction to Atmel Atmega128 Microcontroller'*
http://www.engr.sjsu.edu/bjfurman/courses/ME106/ME106pdf/intro-atmel.pdf
[Date accessed 16/04/09]

[21]. Wikipedia (2009) – *'Maze Solving Algorithm'*
http://en.wikipedia.org/wiki/Maze_solving_algorithm
[Date accessed 16/04/09]

[22]. ROBOTIS (2006) – '*Bioloid Quick Start Assembly Guide'*

BIBLIOGRAPHY

1. ROBOTIS (2006) – *'Bioloid User's Manual'*

2. ROBOTIS (2006) – *'AX-12 User's Manual'*

3. ROBOTIS (2006) – *'AX-S1 User's Manual'*

4. Robot-Savvy. (2009) – *'Bioloid Comprehensive Kit Knowledgebase'*
   http://robosavvy.com/site/index.php?option=com_content&task=view&id=82&Itemid=81
   [Date accessed 15/04/09]

5. Simpson, Michael. (2007) – *'Understanding the AX-12'*
   http://www.antratek.nl/pdf/Understanding-AX12.pdf
   [Date accessed 13/04/09]

APPENDICES

Appendix A – Project Time Plan

Appendix B – Control Table for Dynamixel AX-12 Servo

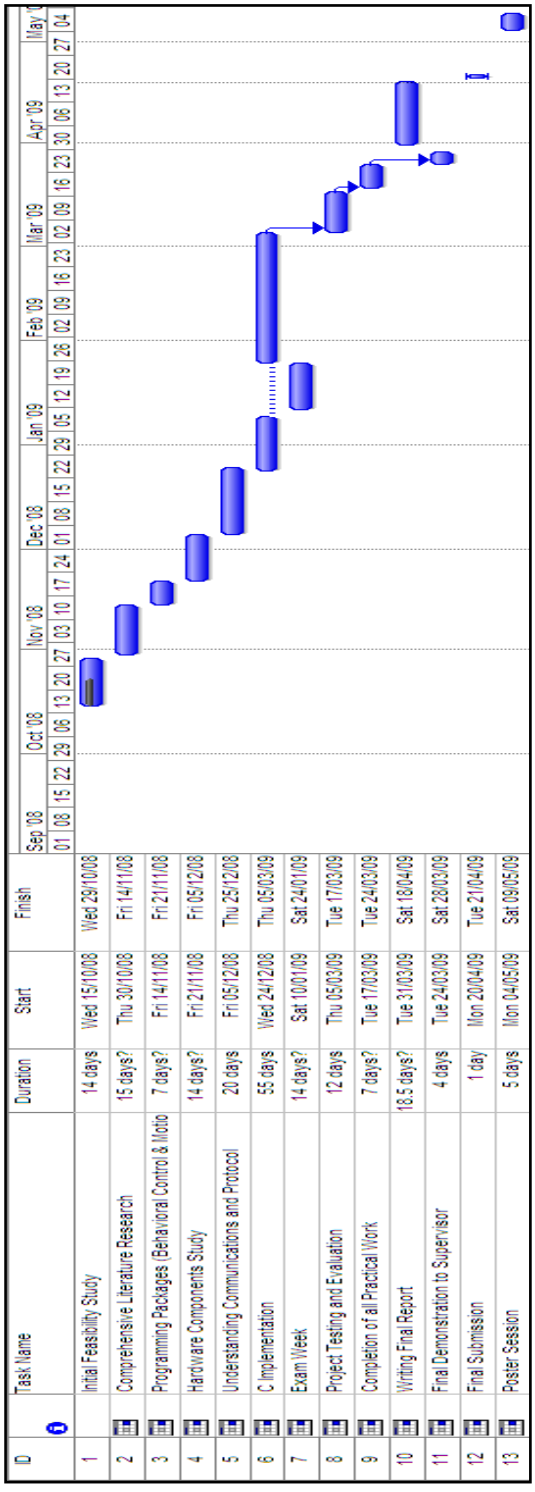
Appendix C – Control Table for Dynamixel AX-S1 Sensor

Appendix D – Instruction Set for Dynamixel

Appendix E – Software Listings

Appendix F – Instruction guide to download a C program on the CM-5

Appendix A – Project Time Plan



ID	Task Name	Duration	Start	Finish
1	Initial Feasibility Study	14 days	Wed 15/10/08	Wed 29/10/08
2	Comprehensive Literature Research	15 days?	Thu 30/10/08	Fri 14/11/08
3	Programming Packages (Behavioral Control & Motio	7 days?	Fri 14/11/08	Fri 21/11/08
4	Hardware Components Study	14 days?	Fri 21/11/08	Fri 05/12/08
5	Understanding Communications and Protocol	20 days	Fri 05/12/08	Thu 25/12/08
6	C Implementation	55 days	Wed 24/12/08	Thu 05/03/09
7	Exam Week	14 days?	Sat 10/01/09	Sat 24/01/09
8	Project Testing and Evaluation	12 days	Thu 05/03/09	Tue 17/03/09
9	Completion of all Practical Work	7 days?	Tue 17/03/09	Tue 24/03/09
10	Writing Final Report	18.5 days?	Tue 31/03/09	Sat 18/04/09
11	Final Demonstration to Supervisor	4 days	Tue 24/03/09	Sat 28/03/09
12	Final Submission	1 day	Mon 20/04/09	Tue 21/04/09
13	Poster Session	5 days	Mon 04/05/09	Sat 09/05/09

Appendix B – AX-12 Servo Control Table

Control Table

	Address	Item	Access	Initial Value
EEPROM Area	0(0X00)	Model Number(L)	RD	12(0x0C)
	1(0X01)	Model Number(H)	RD	0(0x00)
	2(0X02)	Version of Firmware	RD	?
	3(0X03)	ID	RD,WR	1(0x01)
	4(0X04)	Baud Rate	RD,WR	1(0x01)
	5(0X05)	Return Delay Time	RD,WR	250(0xFA)
	6(0X06)	CW Angle Limit(L)	RD,WR	0(0x00)
	7(0X07)	CW Angle Limit(H)	RD,WR	0(0x00)
	8(0X08)	CCW Angle Limit(L)	RD,WR	255(0xFF)
	9(0X09)	CCW Angle Limit(H)	RD,WR	3(0x03)
	10(0x0A)	(Reserved)	-	0(0x00)
	11(0X0B)	the Highest Limit Temperature	RD,WR	85(0x55)
	12(0X0C)	the Lowest Limit Voltage	RD,WR	60(0X3C)
	13(0X0D)	the Highest Limit Voltage	RD,WR	190(0xBE)
	14(0X0E)	Max Torque(L)	RD,WR	255(0XFF)
	15(0X0F)	Max Torque(H)	RD,WR	3(0x03)
	16(0X10)	Status Return Level	RD,WR	2(0x02)
	17(0X11)	Alarm LED	RD,WR	4(0x04)
	18(0X12)	Alarm Shutdown	RD,WR	4(0x04)
	19(0X13)	(Reserved)	RD,WR	0(0x00)
	20(0X14)	Down Calibration(L)	RD	?
	21(0X15)	Down Calibration(H)	RD	?
	22(0X16)	Up Calibration(L)	RD	?
	23(0X17)	Up Calibration(H)	RD	?
RAM Area	24(0X18)	Torque Enable	RD,WR	0(0x00)
	25(0X19)	LED	RD,WR	0(0x00)
	26(0X1A)	CW Compliance Margin	RD,WR	0(0x00)
	27(0X1B)	CCW Compliance Margin	RD,WR	0(0x00)
	28(0X1C)	CW Compliance Slope	RD,WR	32(0x20)
	29(0X1D)	CCW Compliance Slope	RD,WR	32(0x20)
	30(0X1E)	Goal Position(L)	RD,WR	[Addr36]value
	31(0X1F)	Goal Position(H)	RD,WR	[Addr37]value
	32(0X20)	Moving Speed(L)	RD,WR	0
	33(0X21)	Moving Speed(H)	RD,WR	0
	34(0X22)	Torque Limit(L)	RD,WR	[Addr14] value
	35(0X23)	Torque Limit(H)	RD,WR	[Addr15] value
	36(0X24)	Present Position(L)	RD	?
	37(0X25)	Present Position(H)	RD	?
	38(0X26)	Present Speed(L)	RD	?
	39(0X27)	Present Speed(H)	RD	?
	40(0X28)	Present Load(L)	RD	?
	41(0X29)	Present Load(H)	RD	?
	42(0X2A)	Present Voltage	RD	?
	43(0X2B)	Present Temperature	RD	?
	44(0X2C)	Registered Instruction	RD,WR	0(0x00)
	45(0X2D)	(Reserved)	-	0(0x00)
	46[0x2E)	Moving	RD	0(0x00)
	47[0x2F)	Lock	RD,WR	0(0x00)
	48[0x30)	Punch(L)	RD,WR	32(0x20)
	49[0x31)	Punch(H)	RD,WR	0(0x00)

Appendix C – AX-S1 Sensor Control Table

	Address	Item	Access	Initial Value
Control Table	0[0X00]	Model Number[L]	RD	13[0x0D]
	1[0X01]	Model Number[H]	RD	0[0x00]
	2[0X02]	Version of Firmware	RD	?
	3[0X03]	ID	RD,WR	100[0x64]
	4[0X04]	Baud Rate	RD,WR	1[0x01]
	5[0X05]	Return Delay Time	RD,WR	250[0xFA]
	6[0X06]	[Reserved]	RD,WR	255[0xFF]
	7[0X07]	[Reserved]	RD,WR	3[0x03]
	8[0X08]	[Reserved]	RD,WR	255[0xFF]
	9[0X09]	[Reserved]	RD,WR	3[0x03]
	10[0x0A]	[Reserved]	–	0[0x00]
EEPROM Area	11[0X0B]	the Highest Limit Temperature	RD,WR	100[0x64]
	12[0X0C]	the Lowest Limit Voltage	RD,WR	60[0X3C]
	13[0X0D]	the Highest Limit Voltage	RD,WR	190[0xBE]
	14[0X0E]	[Reserved]	RD,WR	255[0XFF]
	15[0X0F]	[Reserved]	RD,WR	3[0x03]
	16[0X10]	Status Return Level	RD,WR	2[0x02]
	17[0X11]	[Reserved]	RD,WR	4[0x04]
	18[0X12]	[Reserved]	RD,WR	4[0x04]
	19[0X13]	[Reserved]	RD,WR	0[0x00]
	20[0X14]	Obstacle Detected Compare Value	RD,WR	32[0x20]
	21[0X15]	Light Detected Compare Value	RD,WR	32[0x20]
	22[0X16]	[Reserved]	RD,WR	32[0x20]
	23[0X17]	[Reserved]	RD	3[0x03]
	24[0X18]	[Reserved]	RD,WR	0[0x00]
	25[0X19]	[Reserved]	RD,WR	0[0x00]
	26[0X1A]	Left IR Sensor Data	RD	?
	27[0X1B]	Center IR Sensor Data	RD	?
	28[0X1C]	Right IR Sensor Data	RD	?
	29[0X1D]	Left Luminosity	RD	?
	30[0X1E]	Center Luminosity	RD	?
	31[0X1F]	Right Luminosity	RD	?
	32[0X20]	Obstacle Detection Flag	RD	?
	33[0X21]	Luminosity Detection Flag	RD	?
	34[0X22]	[Reserved]	RD,WR	0
	35[0X23]	Sound Data	RD,WR	?
	36[0X24]	Sound Data Max Hold	RD,WR	?
	37[0X25]	Sound Detected Count	RD,WR	?
RAM Area	38[0X26]	Sound Detected Time[L]	RD,WR	?
	39[0X27]	Sound Detected Time[H]	RD,WR	?
	40[0X28]	Buzzer Index	RD,WR	?
	41[0X29]	Buzzer Time	RD,WR	?
	42[0X2A]	Present Voltage	RD	?
	43[0X2B]	Present Temperature	RD	?
	44[0X2C]	Registered Instruction	RD,WR	0[0x00]
	45[0X2D]	[Reserved]	–	0[0x00]
	46[0x2E]	IR Remocon Arrived	RD	0[0x00]
	47[0x2F]	Lock	RD,WR	0[0x00]
	48[0x30]	IR Remocon RX Data 0	RD	?
	49[0x31]	IR Remocon RX Data 1	RD	?
	50[0x32]	IR Remocon TX Data 0	RD,WR	?
	51[0x33]	IR Remocon TX Data 1	RD,WR	?
	52[0x34]	Obstacle Detected Compare	RD,WR	?
	53[0x35]	Light Detected Compare	RD,WR	?

Appendix D – Instruction Set for Dynamixel

Instruction	Function	Value	Number of Parameter
PING	No action. Used for obtaining a Status Packet	0x01	0
READ DATA	Reading values in the Control Table	0x02	2
WRITE DATA	Writing values to the Control Table	0x03	2 ~
REG WRITE	Similar to WRITE_DATA, but stays in standby mode until the ACION instruction is given	0x04	2 ~
ACTION	Triggers the action registered by the REG_WRITE instruction	0x05	0
RESET	Changes the control table values of the Dynamixel actuator to the Factory Default Value settings	0x06	0
SYNC WRITE	Used for controlling many Dynamixel actuators at the same time	0x83	4~

Appendix E – Software Listings

```
/*-------------------------------------------------------------------------------
                                    Bioloid.h
                        Acquired from Example.c provided with the Bioloid Kit
              Defines the control table addresses, ports and other hardware
                  dependent items. Also defines the global variables used.
-------------------------------------------------------------------------------*/

#ifndef BIOLOID_H
#define BIOLOID_H

#define cbi(REG8,BITNUM) REG8 &= ~(_BV(BITNUM))
#define sbi(REG8,BITNUM) REG8 |= _BV(BITNUM)


typedef unsigned char byte;
typedef unsigned int word;
#define ON 1
#define OFF 0
#define _ON 0
#define _OFF 1

//--- Control Table Address ---
//EEPROM AREA
#define P_MODEL_NUMBER_L      0
#define P_MODOEL_NUMBER_H     1
#define P_VERSION             2
#define P_ID                  3
#define P_BAUD_RATE           4
#define P_RETURN_DELAY_TIME   5
#define P_CW_ANGLE_LIMIT_L    6
#define P_CW_ANGLE_LIMIT_H    7
#define P_CCW_ANGLE_LIMIT_L   8
#define P_CCW_ANGLE_LIMIT_H   9
#define P_SYSTEM_DATA2        10
#define P_LIMIT_TEMPERATURE   11
#define P_DOWN_LIMIT_VOLTAGE  12
#define P_UP_LIMIT_VOLTAGE    13
#define P_MAX_TORQUE_L        14
#define P_MAX_TORQUE_H        15
#define P_RETURN_LEVEL        16
#define P_ALARM_LED           17
#define P_ALARM_SHUTDOWN      18
#define P_OPERATING_MODE      19
#define P_DOWN_CALIBRATION_L  20
#define P_DOWN_CALIBRATION_H  21
#define P_UP_CALIBRATION_L    22
#define P_UP_CALIBRATION_H    23
```

```
#define P_TORQUE_ENABLE         (24)
#define P_LED                   (25)
#define P_CW_COMPLIANCE_MARGIN  (26)
#define P_CCW_COMPLIANCE_MARGIN (27)
#define P_CW_COMPLIANCE_SLOPE   (28)
#define P_CCW_COMPLIANCE_SLOPE  (29)
#define P_GOAL_POSITION_L       (30)
#define P_GOAL_POSITION_H       (31)
#define P_GOAL_SPEED_L          (32)
#define P_GOAL_SPEED_H          (33)
#define P_TORQUE_LIMIT_L        (34)
#define P_TORQUE_LIMIT_H        (35)
#define P_PRESENT_POSITION_L    (36)
#define P_PRESENT_POSITION_H    (37)
#define P_PRESENT_SPEED_L       (38)
#define P_PRESENT_SPEED_H       (39)
#define P_PRESENT_LOAD_L        (40)
#define P_PRESENT_LOAD_H        (41)
#define P_PRESENT_VOLTAGE       (42)
#define P_PRESENT_TEMPERATURE   (43)
#define P_REGISTERED_INSTRUCTION (44)
#define P_PAUSE_TIME            (45)
#define P_MOVING (46)
#define P_LOCK                  (47)
#define P_PUNCH_L               (48)
#define P_PUNCH_H               (49)

//--- Instruction ---
#define INST_PING           0x01
#define INST_READ           0x02
#define INST_WRITE          0x03
#define INST_REG_WRITE      0x04
#define INST_ACTION         0x05
#define INST_RESET          0x06
#define INST_DIGITAL_RESET  0x07
#define INST_SYSTEM_READ    0x0C
#define INST_SYSTEM_WRITE   0x0D
#define INST_SYNC_WRITE     0x83
#define INST_SYNC_REG_WRITE 0x84

#define CLEAR_BUFFER gbRxBufferReadPointer = gbRxBufferWritePointer
#define DEFAULT_RETURN_PACKET_SIZE 6
#define BROADCASTING_ID 0xfe

#define TxD8 TxD81
#define RxD8 RxD81

//Hardware Dependent Item
#define DEFAULT_BAUD_RATE 34   //57600bps at 16MHz
#define F_CPU 16000000
```

```
// For CM-5
#define RS485_TXD PORTE &= ~_BV(PE3),PORTE |= _BV(PE2)  //_485_DIRECTION = 1
#define RS485_RXD PORTE &= ~_BV(PE2),PORTE |= _BV(PE3) // PORT_485_DIRECTION = 0
#define SET_TxD0_FINISH   sbi(UCSR0A,6)
#define RESET_TXD0_FINISH cbi(UCSR0A,6)
#define CHECK_TXD0_FINISH bit_is_set(UCSR0A,6)
#define SET_TxD1_FINISH  sbi(UCSR1A,6)
#define RESET_TXD1_FINISH cbi(UCSR1A,6)
#define CHECK_TXD1_FINISH bit_is_set(UCSR1A,6)

#define RX_INTERRUPT 0x01
#define TX_INTERRUPT 0x02
#define OVERFLOW_INTERRUPT 0x01
#define SERIAL_PORT0 0
#define SERIAL_PORT1 1
#define BIT_RS485_DIRECTION0  0x08  //Port E
#define BIT_RS485_DIRECTION1  0x04  //Port E

/*Hardware Dependent Item*/
#define TXD1_READY                  bit_is_set(UCSR1A,5) //(UCSR1A_Bit5)
#define TXD1_DATA                   (UDR1)
#define RXD1_READY                  bit_is_set(UCSR1A,7)
#define RXD1_DATA                   (UDR1)

#define TXD0_READY                  bit_is_set(UCSR0A,5)
#define TXD0_DATA                   (UDR0)
#define RXD0_READY                  bit_is_set(UCSR0A,7)
#define RXD0_DATA                   (UDR0)
#define SERVO_NOT_CONNECTED                 999
#define NUM_OF_SERVOS_ATTACHED 19

// --- Global Variable Number ---
volatile byte gbpRxInterruptBuffer[256];
byte gbpParameter[128];
byte gbRxBufferReadPointer;
byte gbpRxBuffer[128];
byte gbpTxBuffer[128];
volatile byte gbRxBufferWritePointer;

#endif BIOLOID_H
```

```
/*---------------------------------------------------------------------
                                    Bioloid.c
       Functions acquired from the Example.c provided with the Bioloid Kit
---------------------------------------------------------------------*

 * included files
 */

#define ENABLE_BIT_DEFINITIONS
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <inttypes.h>
#include <string.h>
#include <stdio.h>

#include "Bioloid.h"

void TxD81(byte bTxdData);
void TxD80(byte bTxdData);
void TxDString(byte *bData);
void TxD8Hex(byte bSentData);
void TxD32Dec(long lLong);
byte RxD81(void);
void PortInitialize(void);
void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt);
byte TxPacket(byte bID, byte bInstruction, byte bParameterLength);
byte RxPacket(byte bRxLength);
void PrintBuffer(byte *bpPrintBuffer, byte bLength);


void PortInitialize(void)
{
  DDRA = DDRB = DDRC = DDRD = DDRE = DDRF = 0; //Set all port to input
  PORTB = PORTC = PORTD = PORTE = PORTF = PORTG = 0x00; //PortData initialize to 0
  cbi(SFIOR,2); //All Port Pull Up ready
  DDRE |= (BIT_RS485_DIRECTION0|BIT_RS485_DIRECTION1); //set output the bit
                                                       //RS485direction
}

/*
TxPacket() send data to RS485.
TxPacket() needs 3 parameter; ID of Dynamixel, Instruction byte, Length of parameters.
TxPacket() return length of Return packet from Dynamixel.
*/
byte TxPacket(byte bID, byte bInstruction, byte bParameterLength)  // Transfers
                                                                   //intstruction packet.
{
    byte bCount,bCheckSum,bPacketLength;
```

```
    gbpTxBuffer[0] = 0xff;  //0xFF 0xFF indicate the start of the instruction packet
    gbpTxBuffer[1] = 0xff;
    gbpTxBuffer[2] = bID;
    gbpTxBuffer[3] = bParameterLength+2; //Length(Paramter,Instruction,Checksum)
    gbpTxBuffer[4] = bInstruction;

    for(bCount = 0; bCount < bParameterLength; bCount++)
    {
        gbpTxBuffer[bCount+5] = gbpParameter[bCount];
    }
    bCheckSum = 0;
    bPacketLength = bParameterLength+4+2;

    for(bCount = 2; bCount < bPacketLength-1; bCount++)
    {
        bCheckSum += gbpTxBuffer[bCount];
    }
    gbpTxBuffer[bCount] = ~bCheckSum; //Writing Checksum with Bit Inversion

    RS485_TXD;
    for(bCount = 0; bCount < bPacketLength; bCount++)
    {
        sbi(UCSR0A,6);//SET_TXD0_FINISH;
        TxD80(gbpTxBuffer[bCount]);
    }
    while(!CHECK_TXD0_FINISH); //Wait until TXD Shift register empty
    RS485_RXD;
    return(bPacketLength);
}

/*
RxPacket() read data from buffer.
RxPacket() need a Parameter; Total length of Return Packet.
RxPacket() return Length of Return Packet.
*/

byte RxPacket(byte bRxPacketLength)
{
#define RX_TIMEOUT_COUNT2   3000L
#define RX_TIMEOUT_COUNT1  (RX_TIMEOUT_COUNT2*10L)
  unsigned long ulCounter;
  byte bCount, bLength, bChecksum;
  byte bTimeout;

  bTimeout = 0;
  for(bCount = 0; bCount < bRxPacketLength; bCount++)
  {
    ulCounter = 0;
    while(gbRxBufferReadPointer == gbRxBufferWritePointer)
    {
      if(ulCounter++ > RX_TIMEOUT_COUNT1)
```

```
      {
        bTimeout = 1;
        break;
      }
    }
    if(bTimeout) break;
    gbpRxBuffer[bCount] = gbpRxInterruptBuffer[gbRxBufferReadPointer++];
  }
  bLength = bCount;
  bChecksum = 0;

  if(gbpTxBuffer[2] != BROADCASTING_ID)
  {
    if(bTimeout && bRxPacketLength != 255)
    {
      TxDString("\r\n [Error:RxD Timeout]");
      CLEAR_BUFFER;
    }

    if(bLength > 3) //checking is available.
    {
      if(gbpRxBuffer[0] != 0xff || gbpRxBuffer[1] != 0xff )
      {
        TxDString("\r\n [Error:Wrong Header]");
        CLEAR_BUFFER;
        return 0;
      }
      if(gbpRxBuffer[2] != gbpTxBuffer[2] )
      {
        TxDString("\r\n [Error:TxID != RxID]");
        CLEAR_BUFFER;
        return 0;
      }
      if(gbpRxBuffer[3] != bLength-4)
      {
        TxDString("\r\n [Error:Wrong Length]");
        CLEAR_BUFFER;
        return 0;
      }
      for(bCount = 2; bCount < bLength; bCount++) bChecksum += gbpRxBuffer[bCount];
      if(bChecksum != 0xff)
      {
        TxDString("\r\n [Error:Wrong CheckSum]");
        CLEAR_BUFFER;
        return 0;
      }
    }
  }
  return bLength;
}
```

```
/*
PrintBuffer() print data in Hex code.
PrintBuffer() needs two parameter; name of Pointer(gbpTxBuffer, gbpRxBuffer)
*/
void PrintBuffer(byte *bpPrintBuffer, byte bLength)
{
    byte bCount;
    for(bCount = 0; bCount < bLength; bCount++)
    {
        TxD8Hex(bpPrintBuffer[bCount]);
        TxD8(' ');
    }
    TxDString("(LEN:");TxD8Hex(bLength);TxD8(')');
}

/*
Print value of Baud Rate.
*/
void PrintBaudrate(void)
{
  TxDString("\r\n RS232:");TxD32Dec((16000000L/8L)/((long)UBRR1L+1L) );
  TxDString(" BPS,");
  TxDString("  RS485:");TxD32Dec((16000000L/8L)/((long)UBRR0L+1L) );
  TxDString(" BPS");
}

/*
SerialInitialize() set Serial Port to initial state.
Vide Mega128 Data sheet about Setting bit of register.
SerialInitialize() needs port, Baud rate, Interrupt value.
*/

void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt)
{
  if(bPort == SERIAL_PORT0)
  {
    UBRR0H = 0; UBRR0L = bBaudrate;
    UCSR0A = 0x02;  UCSR0B = 0x18;
    if(bInterrupt&RX_INTERRUPT) sbi(UCSR0B,7); // RxD interrupt enable
    UCSR0C = 0x06; UDR0 = 0xFF;
    sbi(UCSR0A,6);//SET_TXD0_FINISH; // Note. set 1, then 0 is read
  }
  else if(bPort == SERIAL_PORT1)
  {
    UBRR1H = 0; UBRR1L = bBaudrate;
    UCSR1A = 0x02;  UCSR1B = 0x18;
    if(bInterrupt&RX_INTERRUPT) sbi(UCSR1B,7); // RxD interrupt enable
    UCSR1C = 0x06; UDR1 = 0xFF;
    sbi(UCSR1A,6);//SET_TXD1_FINISH; // Note. set 1, then 0 is read
  }
```

```
}

/*
TxD8Hex() print data seperatly.
ex> 0x1a -> '1' 'a'.
*/
void TxD8Hex(byte bSentData)
{
  byte bTmp;

  bTmp =((byte)(bSentData>>4)&0x0f) + (byte)'0';
  if(bTmp > '9') bTmp += 7;
  TxD8(bTmp);
  bTmp =(byte)(bSentData & 0x0f) + (byte)'0';
  if(bTmp > '9') bTmp += 7;
  TxD8(bTmp);
}

/*
TxD80() send data to USART 0.
*/
void TxD80(byte bTxdData)
{
  while(!TXD0_READY);
  TXD0_DATA = bTxdData;
}

/*
TXD81() send data to USART 1.
*/
void TxD81(byte bTxdData)
{
  while(!TXD1_READY);
  TXD1_DATA = bTxdData;
}

/*
TXD32Dec() change data to decimal number system
*/

void TxD32Dec(long lLong)
{
  byte bCount, bPrinted;
  long lTmp,lDigit;
  bPrinted = 0;
  if(lLong < 0)
  {
    lLong = -lLong;
    TxD8('-');
  }
  lDigit = 1000000000L;
```

```
  for(bCount = 0; bCount < 9; bCount++)
  {
    lTmp = (byte)(lLong/lDigit);
    if(lTmp)
    {
      TxD8(((byte)lTmp)+'0');
      bPrinted = 1;
    }
    else if(bPrinted) TxD8(((byte)lTmp)+'0');
    lLong -= ((long)lTmp)*lDigit;
    lDigit = lDigit/10;
  }
  lTmp = (byte)(lLong/lDigit);
  /*if(lTmp)*/ TxD8(((byte)lTmp)+'0');
}

/*
TxDString() prints data in ACSII code.
*/
void TxDString(byte *bData)
{
  while(*bData)
  {
    TxD8(*bData++);
  }
}

/*
RxD81() read data from UART1.
RxD81() return Read data.
*/
byte RxD81(void)
{
  while(!RXD1_READY);
  return(RXD1_DATA);
}

/*
SIGNAL() UART0 Rx Interrupt - write data to buffer
*/
SIGNAL (SIG_UART0_RECV)
{
  gbpRxInterruptBuffer[(gbRxBufferWritePointer++)] = RXD0_DATA;
}
```

```
/*-------------------------------------------------------------------------
                                    ME.h
        Defines the constants, control table addresses and structures
        for functions to operate the AX-12 servo,read the Motion Editor data and
                            play the motion sequences.
-------------------------------------------------------------------------*/


//#define ENABLE_BIT_DEFINITIONS
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "Bioloid.h"
#include "led.h"
//-------------------------------------------------------------------------

#ifndef ME_H
#define ME_H

//Constants
#define POSE_BASE_ADR                (0x0E000)
#define POSE_OFFSET                  (0x00040)
#define POSE_SIZE                    (0x00040)
#define NO_OF_POSES_PER_PAGE         7

// within a page
#define PAGE_SIZE                    (0x200)
#define PAGE_FAST_FLAG               (0x00010)
#define PAGE_PLAYCOUNT               (0x0000F)
#define PAGE_NUM_OF_POSES            (0x00014)
#define PAGE_POSE_SPEED              (0x00016)
#define PAGE_ACCEL_TIME              (0x00018)
#define PAGE_NEXT_PAGE               (0x00019)
#define PAGE_EXIT_PAGE               (0x0001A)

// Within a pose
#define POSE_PAUSE_ADR               0x0003E
#define POSE_SPEED_ADR               0x0003F
#define SERVO_NOT_CONNECTED          9999


//Structure Definitions
typedef struct
{
        uint16_t CurrentPage;
        uint16_t TotalTime;
        uint16_t PageSpeed;
        uint16_t MoveFinishTime; // Time to wait while move finishes
        uint16_t NoOfPoses;
```

```
        uint16_t CurrentPose;

} MOTION_EDITOR_DATA;

typedef struct
{
        uint8_t PoseSpeed,ServoNo;
        uint32_t PosePause;
        uint32_t PoseAdr;
        uint16_t ServoPos[NUM_OF_SERVOS_ATTACHED];
        uint16_t ServoTarget[NUM_OF_SERVOS_ATTACHED];
} POSE;
#endif ME_H
volatile MOTION_EDITOR_DATA ME;


/*-------------------------------------------------------------------------
                                    ME.c
        Contains functions to operate the AX-12
        for functions to operate the AX-12 servo,read the Motion Editor data and
                                play the motion sequences.
-------------------------------------------------------------------------*/
//#define ENABLE_BIT_DEFINITIONS
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <inttypes.h>
#include <string.h>
#include <stdio.h>
#include <avr/iom128.h>
#define F_CPU 16000000
#include <util/delay.h>
#include "ME.h"

int getCurrentServoPos(uint8_t ServoNo);
void SendServoTargetPos(uint8_t ServoNo, int16_t ThetaTarget,uint16_t OmegaServo);
POSE ReadPose(int Page, char Pose);
int MaxDistance(POSE ReadPose);
uint16_t doPose(int page,char pose);
uint16_t doPage(int page);
void MEInit(void);
void TxD16Hex(uint16_t X);


/*-------------------- ~Gets The Current Position of a Servo~ --------------------*/

int getCurrentServoPos(uint8_t ServoNo)
{
        uint16_t POS_LB=0,POS_HB=0;
        gbpParameter[0] = P_PRESENT_POSITION_L;
```

```
	gbpParameter[1] = 2; //Read Length 2 byte L and H
	TxPacket(ServoNo,INST_READ,2);
	uint8_t bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);

	if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1])
	{
		if(gbpRxBuffer[4]==0)
		{
			POS_LB=gbpRxBuffer[5];
			POS_HB=gbpRxBuffer[6];
		}
		else
		{
			// error
			return SERVO_NOT_CONNECTED;
		}
	}
	else
	{
		//error
		return SERVO_NOT_CONNECTED;
	}

	return (POS_HB<<8) + POS_LB;
}


/*------------------ ~ Sends the Servo to the Target Position ~ ------------------*/

void SendServoTargetPos(uint8_t ServoNo, int16_t Target,uint16_t Speed)
{
	uint8_t servo_value_LB = Target & 0x00FF;
	uint8_t servo_value_HB = (Target & 0xFF00)>>8;

	// send servo value
	gbpParameter[0] = P_GOAL_POSITION_L;
	gbpParameter[1] = servo_value_LB; //Writing Data P_GOAL_POSITION_L
	gbpParameter[2] = servo_value_HB; //Writing Data P_GOAL_POSITION_H
	gbpParameter[3] = (Speed & 0x00FF); //Writing Data P_GOAL_SPEED_L
	gbpParameter[4] = (Speed & 0xFF00)>>8; //Writing Data P_GOAL_SPEED_H

	TxPacket(ServoNo,INST_WRITE,5);
	uint8_t bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);
}

/*--------------------- ~ Initialises Motion Editor Data ~ ----------------------*/
void MEInit(void)
{

	ME.CurrentPage = -1;
	ME.CurrentPose = 0;
```

```
}


/*--------------------- ~ Read Pose Information from the Flash ~ -------------------*/
POSE ReadPose(int Page, char Pose)
{
                POSE Read_Pose;
                uint8_t ServoNo =0;
                for(ServoNo=1;ServoNo < NUM_OF_SERVOS_ATTACHED; ServoNo++)
               // Reads the current positions of all the servos
                {


                        Read_Pose.ServoPos[ServoNo] = getCurrentServoPos(ServoNo);
                }


                cli(); // Disable Interrupts

                //Read the values of the Pose Speed, Page Speed and Pose Pause
        Read_Pose.PosePause = pgm_read_byte_far  ((uint32_t)POSE_BASE_ADR +
        (uint32_t)POSE_OFFSET + (uint32_t)PAGE_SIZE * (uint32_t)Page + (uint32_t)POSE_SIZE *
        (uint32_t)Pose + (uint32_t)POSE_PAUSE_ADR);

        Read_Pose.PoseAdr   =  (uint32_t)POSE_BASE_ADR + (uint32_t)POSE_OFFSET +
        (uint32_t)PAGE_SIZE * (uint32_t)Page + (uint32_t)POSE_SIZE * (uint32_t)Pose;

        Read_Pose.PoseSpeed = pgm_read_byte_far ((uint32_t)POSE_BASE_ADR +
        (uint32_t)POSE_OFFSET + (uint32_t)PAGE_SIZE * (uint32_t)Page + (uint32_t)POSE_SIZE *
        (uint32_t)Pose + (uint32_t)POSE_SPEED_ADR);

        ME.PageSpeed = pgm_read_byte_far ((uint32_t)POSE_BASE_ADR +(uint32_t)PAGE_SIZE *
        (uint32_t)Page + (uint32_t)PAGE_POSE_SPEED);

        sei();

        for(ServoNo=1;ServoNo < NUM_OF_SERVOS_ATTACHED; ServoNo++)
        {
                if(Read_Pose.ServoPos[ServoNo] != SERVO_NOT_CONNECTED)
                {
                        cli();
                        uint8_t  servo_value_LB = pgm_read_byte_far  (Read_Pose.PoseAdr +
                        (uint32_t)ServoNo * 2   );

                        uint16_t servo_value_HB = pgm_read_byte_far  (Read_Pose.PoseAdr +
                        (uint32_t)ServoNo * 2 + 1);
                        sei();


                        uint16_t PageFast = pgm_read_byte_far ((uint32_t)POSE_BASE_ADR
                        +(uint32_t)PAGE_SIZE*(uint32_t)Page + (uint32_t)PAGE_FAST_FLAG);
```

```
                        if((PageFast==10) && (ME.PageSpeed>=10)){ME.PageSpeed=160;} if ((Page
                        == 60) || (Page == 69) || (Page == 87) || (Page == 96) || (Page == 67))

                        Read_Pose.PoseSpeed = 10;
                        if (Page == 1)
                                Read_Pose.PoseSpeed = 10;

                        Read_Pose.ServoTarget[ServoNo] = (servo_value_HB<<8) + servo_value_LB;

                        if (Read_Pose.ServoTarget[ServoNo] > 1020)
                                Read_Pose.ServoTarget[ServoNo] = 1020;

                        }
                }

                return Read_Pose;
}

/*---- ~ Returns the maximum distance a servo has to move in a particular pose ~ ---*/

int MaxDistance(POSE Read_Pose)
{
        uint8_t ServoNo =0;
        int16_t Difference;
        uint16_t Max =0;

        int temp =1;

        for(ServoNo=1;ServoNo < NUM_OF_SERVOS_ATTACHED; ServoNo++)
        {
                Difference = Read_Pose.ServoTarget[ServoNo] - Read_Pose.ServoPos[ServoNo];

                if (Difference<0)
                        Difference = -(Difference);
                if (Difference > Max)
                {
                        Max = Difference;
                        temp  = ServoNo;
                }

        }
        return Max;
}
```

```
/*---- ~ Sends the robot to the servo positions specified in a Pose of the Page ---*/ /*------
-----------------------------passed ~ ----------------------------------*/

uint16_t doPose(int page,char pose)
{
	float NSpeed;
	POSE Read_Pose;
	uint8_t ServoNo = 0;
	uint16_t MaxServoSpeed, ServoRefSpeed;
	int16_t Difference;

	Read_Pose = ReadPose(page, pose);  // Read Pose Information
	int Max = MaxDistance(Read_Pose);  // Calculate the max distance to be moved


/*NSpeed is the Normalised Page Pose Speed. If both Page n Pose Speeds are 32, the speed of
servo is 90 degrees/sec.
Thus NSpeed = 1. (32 x 32 = 1024) */

	NSpeed = ((float)(ME.PageSpeed * Read_Pose.PoseSpeed)) / 1024.0;

	/*Each servo can move a max of 300 degrees. Its position values can range from 0 -
	1023. Thus a change of 3.41 units moves 1 degree.

	Since, Time = Dist/Speed.
	Distance = Max / 3.41
	Speed    = NSpeed * 90 degrees

	Time in Steps = Time * 7.81
	*/

	ME.MoveFinishTime = (uint16_t)(float)((((7.81*Max)/3.41)/(NSpeed * 90)) * ((128
*32)/ME.PageSpeed));


	/*Max Servo Speed is the speed of the Servo that has to move at the most distance.
135 is the speed value stored for the servo to move at 90 degrees/sec.
Will change if NSpeed changes i.e. Pages Speed or Pose Speed changes from default 32*/

	MaxServoSpeed = (NSpeed * 135.0);

	Read_Pose.PosePause = (uint16_t)(float)((Read_Pose.PosePause * 32)/(ME.PageSpeed));


	for(ServoNo=1;ServoNo < NUM_OF_SERVOS_ATTACHED; ServoNo++)
	{
		Difference = Read_Pose.ServoTarget[ServoNo] - Read_Pose.ServoPos[ServoNo];

		if (Difference<0)
		{
			Difference = -(Difference);
```

```
		}

/*Calculate the speed of each servo in reference to the Servo Moving at the Max speed. (one
that has to move the max distance.) */

	ServoRefSpeed = (uint16_t)((float) MaxServoSpeed * ((float)Difference/(float)Max));

	if (ServoRefSpeed>MaxServoSpeed)
		ServoRefSpeed = MaxServoSpeed;

	SendServoTargetPos(ServoNo,Read_Pose.ServoTarget[ServoNo],ServoRefSpeed);
	Read_Pose.ServoPos[ServoNo] = Read_Pose.ServoTarget[ServoNo];
	}
	_delay_ms(ME.MoveFinishTime + 10);

	return (Read_Pose.PosePause);
}


/*----- ~ Implements the Motion poses of the Motion page passed in a sequence ~ ----*/

uint16_t doPage(int page)
{
	if(page != -1)
	{
		// New request
		ME.CurrentPage = page;

		cli();
		ME.NoOfPoses= pgm_read_byte_far ((uint32_t)POSE_BASE_ADR + (uint32_t)PAGE_SIZE
		* (uint32_t)ME.CurrentPage + (uint32_t)PAGE_NUM_OF_POSES);
		sei();

		ME.TotalTime = 0;
		ME.CurrentPose = 0;

		do
		{
			_delay_ms(doPose(ME.CurrentPage, ME.CurrentPose));
		ME.CurrentPose++;
		} while (ME.CurrentPose < ME.NoOfPoses);


		return ME.TotalTime;
	}
	return 9999;
}
```

```
/*---------- ~ Implements the Motion Pages in a sequence forming a motion~ --------*/

void doMove(int page)
{

        uint8_t NextPage = page;
        do
        {
                ME.CurrentPage = NextPage;
                cli();
                NextPage= pgm_read_byte_far ((uint32_t)POSE_BASE_ADR +
                (uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage + (uint32_t)PAGE_NEXT_PAGE);
                sei();
                _delay_ms(doPage(ME.CurrentPage));
        } while (NextPage!=0);
}

/*------------------------ ~ Prints the passed hex value~ ----------------------*/

void TxD16Hex(uint16_t X)
{
        uint8_t Lower;
        uint8_t Higher;

        Lower = (X & 0x00FF);
        Higher= (X & 0xFF00)>>8;

        TxD8Hex(Higher);
        TxD8Hex(Lower);

}
```

```
/*-------------------------------------------------------------------------
                            Sensor.h
              Defines the constants and control table addresses for the AX-S1
              Sensor Module. Also contains the function prototypes of the functions
                            designed to operate the sensor.
-------------------------------------------------------------------------*/
#ifndef SENSOR_H
#define SENSOR_H
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "Bioloid.h"
#include "led.h"

#define SensorID                                100

//--- Control Table Address ---
//EEPROM AREA


#define S_MODEL_NUMBER_L                        0
#define S_MODOEL_NUMBER_H                       1
#define S_VERSION                               2
#define S_ID                                    3
#define S_BAUD_RATE                             4
#define S_RETURN_DELAY_TIME                     5
#define S_LIMIT_TEMPERATURE                     11
#define S_DOWN_LIMIT_VOLTAGE                    12
#define S_UP_LIMIT_VOLTAGE                      13
#define S_RETURN_LEVEL                          16
#define S_OBSTACLE_DETECTED_COMPARE_VALUE       20
#define S_LIGHT_DETECTED_COMPARE_VALUE          21


#define S_LEFT_IR_SENSOR_DATA                   (26)
#define S_CENTER_IR_SENSOR_DATA                 (27)
#define S_RIGHT_IR_SENSOR_DATA                  (28)
#define S_LEFT_LUMINOSITY                       (29)
#define S_CENTER_LUMINOSITY                     (30)
#define S_RIGHT_LUMINOSITY                      (31)
#define S_OBSTACLE_DETECTION_FLAG               (32)
#define S_LUMINOSITY_DETECTION_FLAG             (33)
#define S_SOUND_DATA                            (35)
#define S_SOUND_DATA_MAX_HOLD                   (36)
#define S_SOUND_DETECTED_COUNT                  (37)
#define S_SOUND_DETECTED_TIME_L                 (38)
#define S_SOUND_DETECTED_TIME_H                 (39)
#define S_BUZZER_INDEX                          (40)
#define S_BUZZER_TIME                           (41)
#define S_PRESENT_VOLTAGE                       (42)
```

```
#define S_PRESENT_TEMPERATURE                   (43)
#define S_REGISTERED_INSTRUCTION                (44)
#define S_IR_REMCON_ARRIVED                     (46)
#define S_LOCK                                  (47)
#define S_IR_REMCON_TX_DATA0                    (48)
#define S_IR_REMCON_RX_DATA1                    (49)
#define S_IR_REMCON_TX_DATA0                    (50)
#define S_IR_REMCON_TX_DATA1                    (51)
#define S_OBSTACLE_DETECTED_COMPARE             (52)
#define S_LIGHT_DETECTED_COMPARE                (53)

#define S_LEFT                                  1
#define S_FRONT                                 2
#define S_FRONT_LEFT                            3
#define S_RIGHT                                 4
#define S_LEFT_RIGHT                            5
#define S_FRONT_RIGHT                           6
#define S_FRONT_LEFT_RIGHT                      7

#endif SENSOR_H
```

```
/*-------------------------------------------------------------------------------
                              Sensor.c
               Contains the implementation of functions written to operate
                              the AX-S1 sensor
-------------------------------------------------------------------------------*//*
 * included files
 */
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <inttypes.h>
#include <string.h>
#include <stdio.h>


#include<avr/iom128.h>
#define F_CPU 16000000
#include <util/delay.h>

#include "Sensor.h"

#define LEFT_OBSTACLE_THRESHOLD             7
#define RIGHT_OBSTACLE_THRESHOLD            7
#define CENTER_OBSTACLE_THRESHOLD           35


void SensorInit()
{
        uint8_t Compare_Value = 60;


        // send servo value
        gbpParameter[0] = S_OBSTACLE_DETECTED_COMPARE;
        gbpParameter[1] = Compare_Value;

        TxPacket(SensorID,INST_WRITE,2);

}

/*--------- ~ Checks left sensor. Returns 1 if an obstacle is detected ~ ----------*/
uint8_t CheckLeftSensor()
{
        uint8_t LeftObstacle;

        gbpParameter[0] = S_LEFT_IR_SENSOR_DATA;
        gbpParameter[1] = 1; //Read Compare Bit

        TxPacket(SensorID,INST_READ,2);
        uint8_t bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
```

```
	if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE + gbpParameter[1])
	{
		if(gbpRxBuffer[4]==0)
		{
			LeftObstacle = gbpRxBuffer[5];
		}
		else
		{
			// error
			return SERVO_NOT_CONNECTED;
		}
	}
	else
	{
		//error
		return SERVO_NOT_CONNECTED;
	}

	if (LeftObstacle > LEFT_OBSTACLE_THRESHOLD)
		return 1;
	else
		return 0;

}


/*--------- ~ Checks center sensor. Returns 1 if an obstacle is detected ~ --------*/
uint8_t CheckCenterSensor()
{
	uint8_t CenterObstacle;

	gbpParameter[0] = S_CENTER_IR_SENSOR_DATA;
	gbpParameter[1] = 1; //Read Compare Bit

	TxPacket(SensorID,INST_READ,2);

	uint8_t bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);

	if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE + gbpParameter[1])
	{
		if(gbpRxBuffer[4]==0)
		{
			CenterObstacle = gbpRxBuffer[5];
		}
		else
		{
			// error
			return SERVO_NOT_CONNECTED;
		}
	}
```

```
	else
	{
		//error
		return SERVO_NOT_CONNECTED;
	}

	if (CenterObstacle > CENTER_OBSTACLE_THRESHOLD)
		return 1;
	else
		return 0;

}

/*------- ~ Checks right sensor. Returns 1 if an obstacle is detected ~ ----------*/
uint8_t CheckRightSensor()
{
	uint8_t RightObstacle;

	gbpParameter[0] = S_RIGHT_IR_SENSOR_DATA;
	gbpParameter[1] = 1; //Read Compare Bit

	TxPacket(SensorID,INST_READ,2);
	uint8_t bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);

	if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE + gbpParameter[1])
	{
		if(gbpRxBuffer[4]==0)
		{
			RightObstacle = gbpRxBuffer[5];
		}
		else
		{
			// error
			return SERVO_NOT_CONNECTED;
		}
	}
	else
	{
		//error
		return SERVO_NOT_CONNECTED;
	}

	if (RightObstacle > RIGHT_OBSTACLE_THRESHOLD)
		return 1;
	else
		return 0;

}
```

```
/*------------ ~ Checks the sensors and returns a 3 bit value. ~ ------------------*/

uint8_t CheckSensor()
{

	uint8_t Obstacle_Detection_Flag;

	/*Bit 0 - Object detected on left
	  Bit 1 - Object detected on center
	  Bit 3 - Object detected on right */

	gbpParameter[0] = S_OBSTACLE_DETECTION_FLAG;
	gbpParameter[1] = 1; //Read Compare Bit

	TxPacket(SensorID,INST_READ,2);
	uint8_t bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);

	if(bRxPacketLength == DEFAULT_RETURN_PACKET_SIZE + gbpParameter[1])
	{
		if(gbpRxBuffer[4]==0)
		{
			Obstacle_Detection_Flag = gbpRxBuffer[5];
		}
		else
		{
			// error
			return SERVO_NOT_CONNECTED;
		}
	}
	else
	{
		//error
		return SERVO_NOT_CONNECTED;
	}

	return Obstacle_Detection_Flag;
}
```

```
/*****************************************************************************
                                    Main.c
                            Maze Solving by a Humanoid Robot
                                    Nauman Shah
                                    07155081
                                Final Year Project
                            University of Hertfordshire
*****************************************************************************/

/*
 * included files
 */

#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "ME.h"
#include "Sensor.c"


/*Hardware Dependent Item*/
#define TXD1_READY                  bit_is_set(UCSR1A,5) //(UCSR1A_Bit5)
#define TXD1_DATA                   (UDR1)
#define RXD1_READY                  bit_is_set(UCSR1A,7)
#define RXD1_DATA                   (UDR1)

#define TXD0_READY                  bit_is_set(UCSR0A,5)
#define TXD0_DATA                   (UDR0)
#define RXD0_READY                  bit_is_set(UCSR0A,7)
#define RXD0_DATA                   (UDR0)
#define SERVO_NOT_CONNECTED         9999
#define NUM_OF_SERVOS_ATTACHED      19

int main(void)
{
  PortInitialize(); //Port In/Out Direction Definition



/*** Initialisation****/

 RS485_RXD; //Set RS485 Direction to Input State.
 SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);//RS485 Initializing(RxInterrupt)
 SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,0); //RS232 Initializing(None
                                                     //Interrupt)

 SensorInit();

 gbRxBufferReadPointer = gbRxBufferWritePointer = 0;  //RS485 RxBuffer Clearing.
```

```
 sei();  //Enable Interrupt -- Compiler Function

 doPage(1);
 _delay_ms(1000);
 doPage(25);
 _delay_ms(1000);
 WallFollowing();

 }
int WallFollowing()
{
        uint8_t NextPage;
        uint8_t ExitPage;
        uint8_t CheckLeftFlag;
        uint8_t CheckCenterFlag;
        uint8_t CheckRightFlag;
        int i=0;
        int j=0;
        doPage(1);
        _delay_ms(1000);

        while (1)
        {
                CheckLeftFlag = CheckLeftSensor();           // Check left sensor
                CheckCenterFlag = CheckCenterSensor();       // Check center sensor
                CheckRightFlag = CheckRightSensor();         // Check right sensor

                //--------Walk Straight------------//
                NextPage = 60;
                if (((CheckLeftFlag == 0) && (CheckCenterFlag == 0) && (CheckRightFlag == 1))
                || ((CheckLeftFlag == 1) && (CheckCenterFlag == 0) && (CheckRightFlag == 1)))
                {
                        while (((CheckLeftFlag == 0) && (CheckCenterFlag == 0) &&
                        (CheckRightFlag == 1)) || ((CheckLeftFlag == 1) && (CheckCenterFlag ==
                        0) && (CheckRightFlag == 1)))
                        {
                                        ME.CurrentPage = NextPage;

                                        cli();
                                        NextPage= pgm_read_byte_far ((uint32_t)POSE_BASE_ADR +
                                        (uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage +
                                        (uint32_t)PAGE_NEXT_PAGE);
                                        sei();

                                        doPage(ME.CurrentPage);
                                        _delay_ms(2);

                                        CheckLeftFlag = CheckLeftSensor();
                                        CheckCenterFlag = CheckCenterSensor();
                                        CheckRightFlag = CheckRightSensor();
                        }
```

```
                                ExitPage = pgm_read_byte_far  ((uint32_t)POSE_BASE_ADR +
                                (uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage +
                                (uint32_t)PAGE_EXIT_PAGE);
                                _delay_ms(doPage(ExitPage));
                                _delay_ms(500);
                                doPage(1);
                                _delay_ms(1000);
                        }

                //---------Turn Right------------//
                if (((CheckLeftFlag == 0) && (CheckCenterFlag == 0) && (CheckRightFlag == 0))
|| ((CheckLeftFlag == 0) && (CheckCenterFlag == 1) && (CheckRightFlag == 0))  ||
((CheckLeftFlag == 1) && (CheckCenterFlag == 1) && (CheckRightFlag == 0)) ||  ((CheckLeftFlag
== 1) && (CheckCenterFlag == 0) && (CheckRightFlag == 0)))
                {

                        for (i=0;i<2;i++)
                        {
                                NextPage = 87;
                                for(j=0;j<4;j++) // The 4 pages of Right Turn
                                {
                                        ME.CurrentPage = NextPage;

                                        cli();
                                        NextPage= pgm_read_byte_far ((uint32_t)POSE_BASE_ADR +
                                        (uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage +
                                        (uint32_t)PAGE_NEXT_PAGE);
                                        sei();

                                        doPage(ME.CurrentPage);

                                }
                                _delay_ms(500);
                        }
                        doPage(1);
                        _delay_ms(500);

                        NextPage = 60;
                        for (i=0; i<7;i++)
                        {
                                        ME.CurrentPage = NextPage;

                                        cli();
                                        NextPage= pgm_read_byte_far ((uint32_t)POSE_BASE_ADR +
                                        (uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage +
                                        (uint32_t)PAGE_NEXT_PAGE);
                                        sei();

                                        doPage(ME.CurrentPage);
                                        _delay_ms(2);
```

```
                        }

                        ExitPage = pgm_read_byte_far  ((uint32_t)POSE_BASE_ADR +
(uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage + (uint32_t)PAGE_EXIT_PAGE);
                        _delay_ms(doPage(ExitPage));
                        _delay_ms(500);
                        doPage(1);
                        _delay_ms(1000);

                }

                //------------Turn Left-----------//
                if (((CheckLeftFlag == 0) && (CheckCenterFlag == 1) && (CheckRightFlag == 1))
                || ((CheckLeftFlag == 1) && (CheckCenterFlag == 1)  && (CheckRightFlag == 1)))
                {
                        for (i=0;i<2;i++)
                        {
                                NextPage = 96;

                                for(j=0;j<4;j++) // The 4 pages of Left Turn
                                {
                                        ME.CurrentPage = NextPage;

                                        cli();
                                        NextPage= pgm_read_byte_far ((uint32_t)POSE_BASE_ADR +
                                        (uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage +
                                        (uint32_t)PAGE_NEXT_PAGE);
                                        sei();

                                        doPage(ME.CurrentPage);

                                }
                                _delay_ms(500);
                        }
                        doPage(1);
                        _delay_ms(500);
                }

                if ((CheckLeftFlag ==  1) && (CheckCenterFlag == 0) && (CheckRightFlag == 0))
                {
                        NextPage = 87;
                        for(j=0;j<4;j++) // The 4 pages of Right Turn
                        {
                                ME.CurrentPage = NextPage;

                                cli();
                                NextPage= pgm_read_byte_far ((uint32_t)POSE_BASE_ADR +
(uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage + (uint32_t)PAGE_NEXT_PAGE);
                                sei();
```

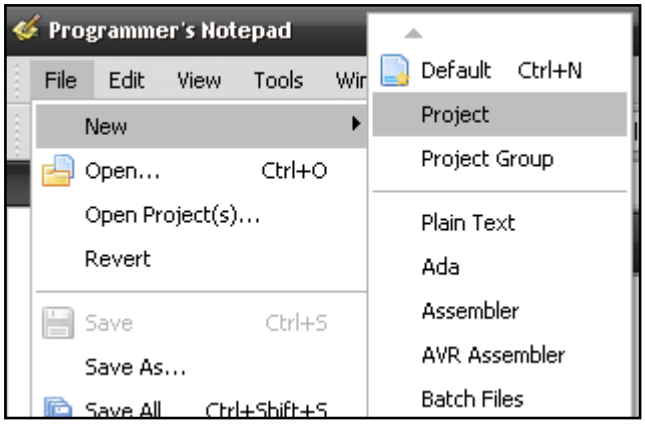
```
                        doPage(ME.CurrentPage);

                }
                doPage(1);
                _delay_ms(500);

                NextPage = 60;
                for (int i=0; i<7;i++)
                {
                        ME.CurrentPage = NextPage;

                        cli();
                        NextPage= pgm_read_byte_far ((uint32_t)POSE_BASE_ADR +
(uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage + (uint32_t)PAGE_NEXT_PAGE);
                        sei();

                        doPage(ME.CurrentPage);
                        _delay_ms(2);

                        CheckLeftFlag = CheckLeftSensor();
                        CheckCenterFlag = CheckCenterSensor();
                        CheckRightFlag = CheckRightSensor();
                }

                ExitPage = pgm_read_byte_far  ((uint32_t)POSE_BASE_ADR +
(uint32_t)PAGE_SIZE*(uint32_t)ME.CurrentPage + (uint32_t)PAGE_EXIT_PAGE);
                _delay_ms(doPage(ExitPage));
                _delay_ms(500);
                doPage(1);
                _delay_ms(1000);
        }

    }
}
```

Appendix F – Instruction guide to download a C program on the CM-5

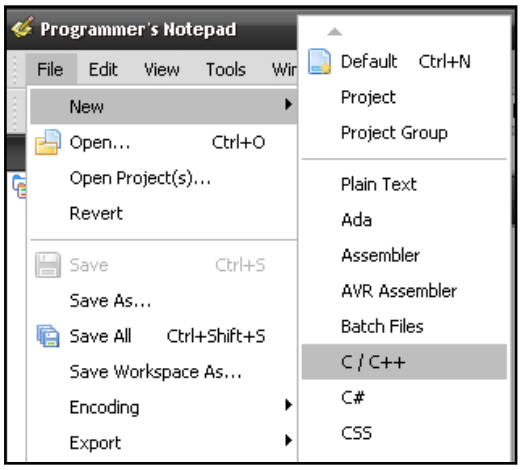
The microcontroller on the CM-5 module of the robot is based on the Atmel Atmega128 microprocessor of the AVR series. Although many different compilers are available to program the microprocessor, the compiler used for this project was a freeware GNU GCC compiler. The GCC compiler has a command line interface, thus, a separate editor is needed to develop the code. WinAVR, by AVRFreaks, is a freeware editor available for download on the internet, coupled with the Programmer's Notebook provides an easy to use platform for developing and compiling a C program. The WinAVR package can be downloaded from the http://winavr.sourceforge.net/ website. Once the package is downloaded and installed, the following instructions can be followed to develop and download a C program to the Bioloid controller.

From the WinAVR menu, select the Programmer's Notebook [WinAVR] to open the editor, as shown below. This editor allows the user to create and manage Projects, which contain all the source file associated with the program being developed.

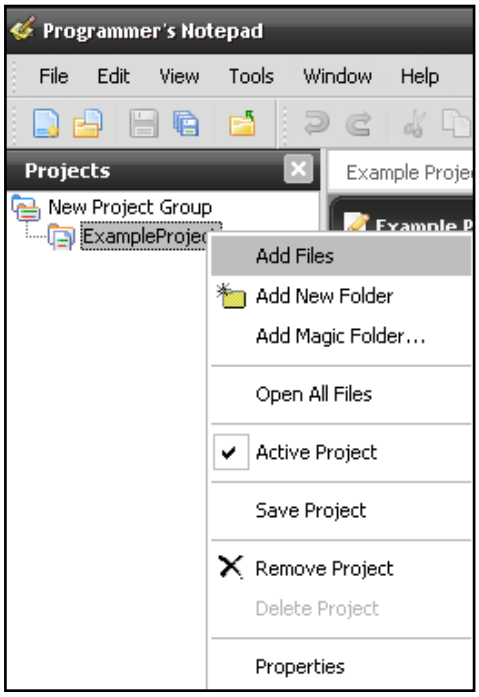
To create a new project, select the 'New' in the 'File' menu of the editor. Select 'Project' to create a new project, or other desired file type.



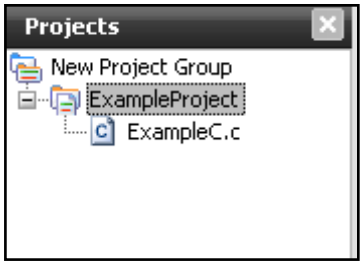
Give the project an appropriate name and specify the location of the Folder to save the project file in. After a project is created, a new C file is to be created to start writing the program. To do so, go to the 'New' menu in the 'File' menu, as done before. Then select 'C/C++' option from the menu, which will open a new editor window.



Save the file with an appropriate name by selecting the 'Save As' option in the 'File' menu. Once the file is file is saved, it can be added to the current project by right-clicking on the project name in the Projects Panel on the left and selecting the 'Add New' option.

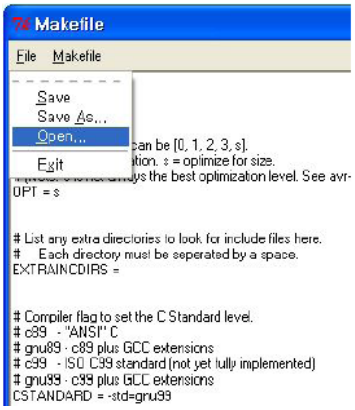


Browse to the created C file and select Open. The added file will now show in the Projects Panel under the name of the project. The program code can now be written in the window. Other source and header files can be added similarly, by clicking the project name in the Projects Panel.



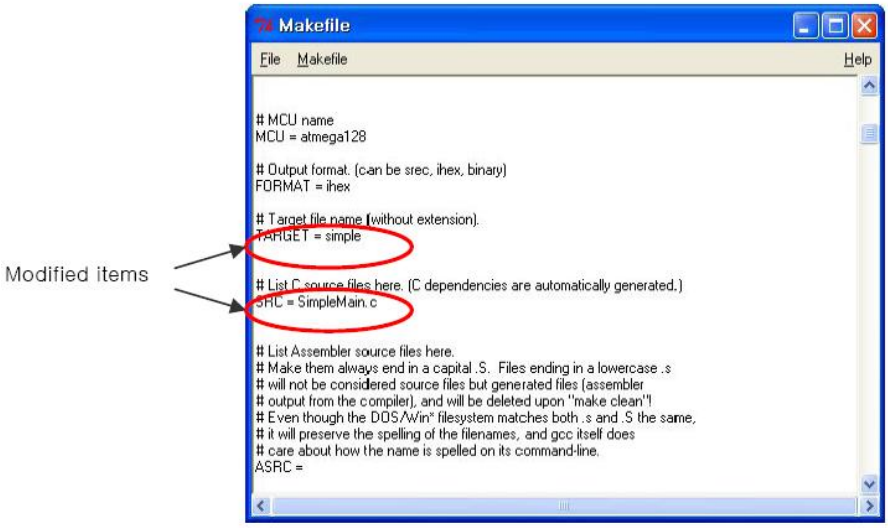
Once the C program is written, it needs to be compiled. Since the WinAVR does not has an internal compiler, it uses a separate 'Makefile' to compile the source code. The 'Makefile' contains the necessary information to derive a target program from a number of sources it is depended upon. The 'Makefile' needs to be in the same folder as the project and the source files, thus first a Makefile is to be copied in the project folder before the program can be compiled. The Makefile can be found in the

CD provided with the Bioloid Kit, in the Examples\CM5\makefile directory. Once this file is copied in the same folder, run the 'mfile' program from the WinAVR menu.

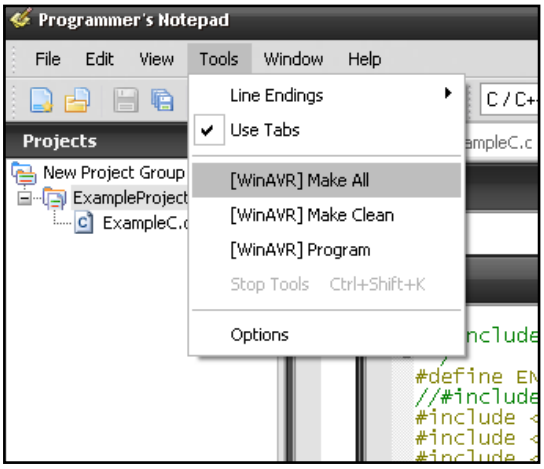

Makefile
File Makefile
Save
Save As...
Open...
Exit
OPT = s
# List any extra directories to look for include files here.
# Each directory must be seperated by a space.
EXTRAINCDIRS =
# Compiler flag to set the C Standard level.
# c89 - "ANSI" C
# gnu89 - c89 plus GCC extensions
# c99 - ISO C99 standard (not yet fully implemented)
# gnu99 - c99 plus GCC extensions
CSTANDARD = -std=gnu99

This program is used to edit the Makefile. Thus, to edit it, open the Makefile by selecting Open option in the File menu and browse to the Makefile.

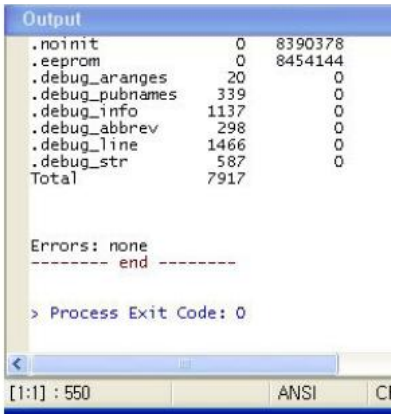
Once the file is open, select the option 'Enable Editing of the Makefile' in the Makefile menu on the top. This allows the Makefile to be edited. Once the option is selected, two values needed to be changed. Enter the name of the resultant hex file to be created next to 'TARGET' and enter the names of all the source files next to 'SRC', as shown in the figure below.


Modified items
Makefile
File Makefile Help
# MCU name
MCU = atmega128
# Output format. (can be srec, ihex, binary)
FORMAT = ihex
# Target file name (without extension).
TARGET = simple
# List C source files here. (C dependencies are automatically generated.)
SRC = SimpleMain.c
# List Assembler source files here.
# Make them always end in a capital .S. Files ending in a lowercase .s
# will not be considered source files but generated files (assembler
# output from the compiler), and will be deleted upon "make clean"!
# Even though the DOS/Win* filesystem matches both .s and .S the same,
# it will preserve the spelling of the filenames, and gcc itself does
# care about how the name is spelled on its command-line.
ASRC =

Once the both the values are entered, click 'Save' from the File menu. Open the 'Programmer's Notepad' and select the 'Make All [WinAVR]' option from the 'Tools' menu to compile the developed code.



Once the code is compiled, the output should display 'Errors: none'. If there are errors, the code needs to be compiled again after removal of the errors.



Once, the code is compiled, a hex file will be generated in the same folder as the project. This hex file can be downloaded on to the CM-5 module using the Robot Terminal. To download, open the Robot Terminal and connect it. Once connected, hold down the '#' key (shift + 3) and turn the CM-5 module on and press enter. Type in the command 'load' and press enter again. Now choose the 'Transfer File' from the 'Files' menu and open the hex file. Type in 'go' in the robot terminal and the file will be downloaded. Press the 'start' button on the CM-5 module and the program will run on the CM-5.